Algorithmique et Programmation

Cours d'introduction à Java (Processing)

IUT Informatique de Bordeaux

Algorithmique et Programmation - S1

- Objectif : Proposer une solution logicielle conforme à un cahier des charges
- Compétences :
  - Écrire un algorithme
  - Utiliser un langage de programmation
  - Compiler, corriger, tester un programme
  - Rendre compte de ses choix de programmation

Plan du cours

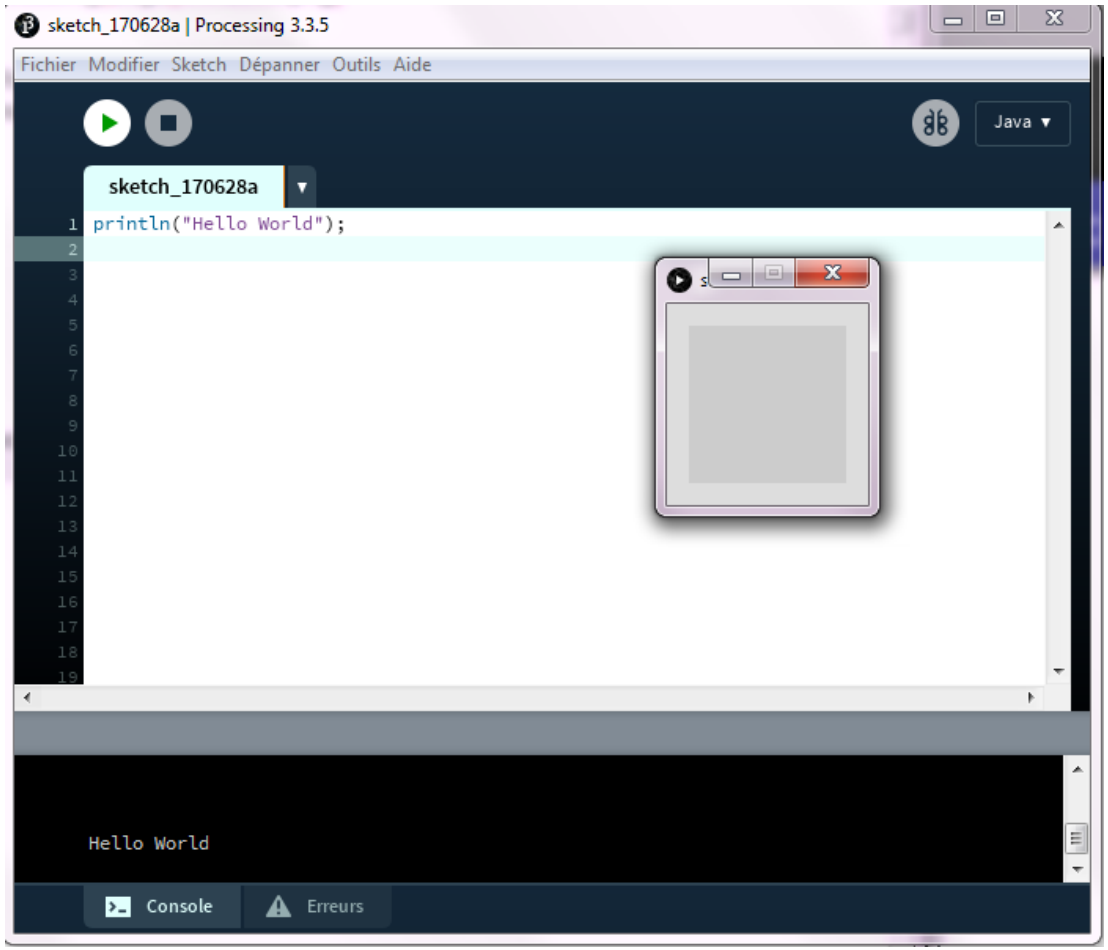
1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

Processing

- Processing a été créé en 2001 par Benjamin Fry et Casey Reas, deux artistes américains.
- Processing est une bibliothèque java et un environnement de développement libre (sous licence GNU GPL).

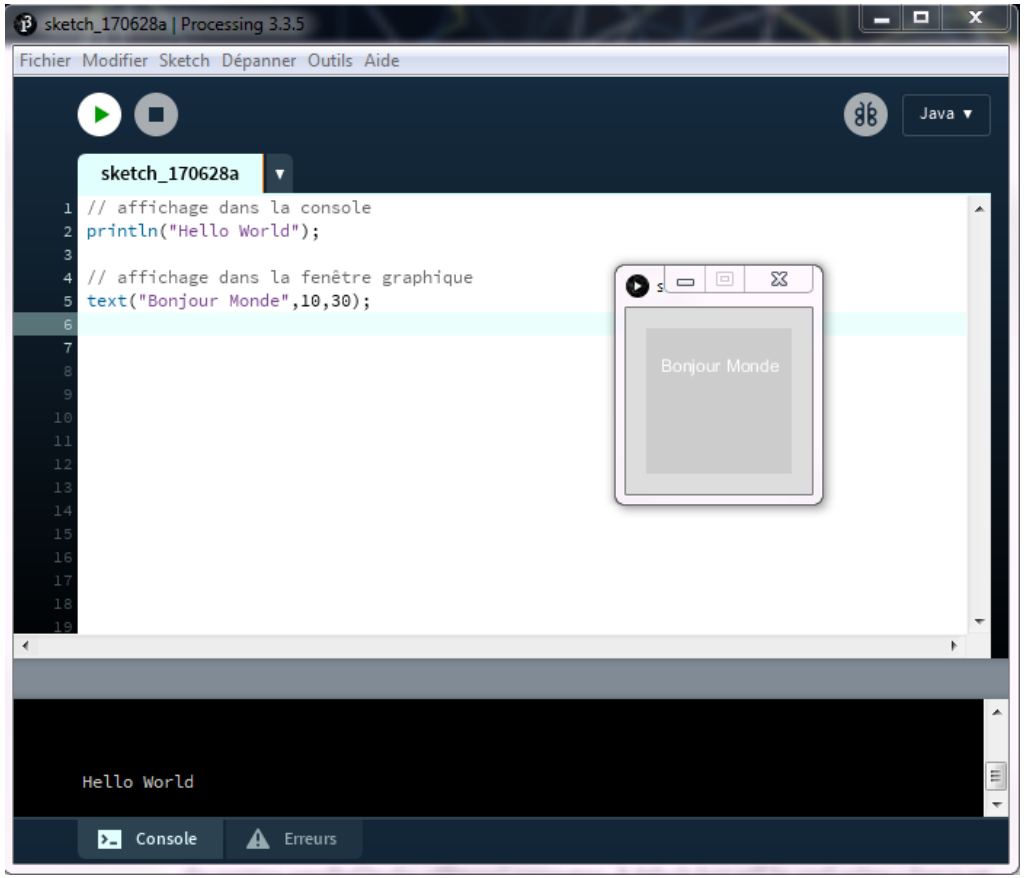
Processing : "Hello World"

```
1  println("Hello World");
```



Processing : "Hello World"

```
1  // affichage dans la console
2  println("Hello World");
3  // affichage dans la fenetre graphique
4  text("Bonjour Monde", 10, 30);
```



Processing : Configuration et Sauvegarde

Configuration

- Fichier/Préférences...
  - L'emplacement du sketchbook peut être modifié
  - Activez l'**Autocomplétion du code (Ctrl+Espace)**

Sauvegarde

- Sauvez régulièrement votre sketch, **il n'y a pas d'auto-save !**
  - La première fois : Fichier/Enregistrer sous...
  - Puis ensuite : Fichier/Enregistrer (Ctrl+S)

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

À vous de jouer !

Coordonnées en pixel

Modifiez le programme précédent :

1. Changez les coordonnées d'affichage du texte dans la fenêtre graphique :
   - en (0,10)
   - en (0,0) : que remarquez-vous ?
2. Affichez le texte suivant dans la fenêtre graphique :

```
Bonjour
tout le monde
ça va bien ?
```

À vous de jouer !

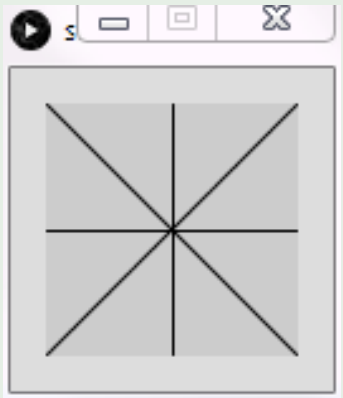
point() et line()

Essayez :

```
1 point(50,30);
2 line(0,0,99,99);
```

Par défaut, la fenêtre est de taille 100 (width) sur 100 (height)

1. Consultez la documentation de line() (Ctrl+Maj+F)
2. Réalisez l'affichage suivant :



À vous de jouer !

stroke()

Essayez :

```
1  stroke(255,0,0);   // ?? dire ici ce que ca fait
2  point(50,30);
3  line(0,0,99,99);
```

Consultez la doc de stroke() et complétez le commentaire

strokeWeight()

Idem avec :

```
1  strokeWeight(5);   // ?? dire ici ce que ca fait
2  point(50,30);
3  line(0,0,99,99);
```

À vous de jouer !

Ligne horizontale centrée (fenêtre 100x100)

```
1 line(0,50,99,50);
```

Ligne "aléatoire" du bord gauche au bord droit

Distance aléatoire du haut de la fenêtre (grâce à random(100))

```
1 line(0, ? ,99, ? );
```

Ligne "aléatoire" horizontale du bord gauche au bord droit

```
1 ??
```

Réponse

Ligne "aléatoire" horizontale du bord gauche au bord droit

```
1  ??
```

Introduction d'une variable

- On veut la même distance du haut pour les 2 bords
- On a donc besoin de **stocker** la valeur retournée par random(100) dans une **variable**

```
float distanceTop = random(100);

line(0,distanceTop,99,distanceTop);
```

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

Qu'est-ce qu'une variable ?

Définition

Une variable est une entité qui contient une information :

Une variable possède :

- un nom qui permet d'y accéder
- un type qui caractérise l'ensemble des valeurs que peut prendre la variable
- une valeur qui peut changer dans le temps

Qu'est-ce qu'une variable ?

Définition

Une variable est une entité qui contient une information :

Une variable possède :

- un nom qui permet d'y accéder
- un type qui caractérise l'ensemble des valeurs que peut prendre la variable
- une valeur qui peut changer dans le temps

Qu'est-ce qu'une variable ?

Définition

Une variable est une entité qui contient une information :

Une variable possède :

- un nom qui permet d'y accéder
- un type qui caractérise l'ensemble des valeurs que peut prendre la variable
- une valeur qui peut changer dans le temps

Qu'est-ce qu'un type ?

Exemple de types associés à des variables :

- âge : Entier (age prend ses valeurs dans $\mathbb{Z}$)
- nbEnfants : Entier
- poids : Réel (poids prend ses valeurs dans $\mathbb{R}$)
- note : Réel
- nom : Chaîne de caractères (succession de caractères)
- adresse : Chaîne de caractères

Qu'est-ce qu'un type ?

Attention

Une fois qu'un type de données est associé à une variable :

- cette variable ne peut plus en changer, et
- le contenu de cette variable doit obligatoirement être du même type.

Exemple

Si on définit : a : **entier** et b : **réel**
On peut :

- Donner à a les valeurs suivantes : {0 ; 2 ; -1 ; 1000 ; ...}
- Donner à b les valeurs suivantes : {0.5 ; 2.17 ; -1.0 ; ...}

MAIS PAS LE CONTRAIRE !

Qu'est-ce qu'un type ?

Attention

Une fois qu'un type de données est associé à une variable :

- cette variable ne peut plus en changer, et
- le contenu de cette variable doit obligatoirement être du même type.

Exemple

Si on définit : a : **entier** et b : **réel**

On peut :

- Donner à a les valeurs suivantes : {0 ; 2 ; -1 ; 1000 ; ...}
- Donner à b les valeurs suivantes : {0.5 ; 2.17 ; -1.0 ; ...}

MAIS PAS LE CONTRAIRE !

Qu'est-ce qu'un type ?

Attention

Une fois qu'un type de données est associé à une variable :

- cette variable ne peut plus en changer, et
- le contenu de cette variable doit obligatoirement être du même type.

Exemple

Si on définit : a : **entier** et b : **réel**

On peut :

- Donner à a les valeurs suivantes : {0 ; 2 ; -1 ; 1000 ; ...}
- Donner à b les valeurs suivantes : {0.5 ; 2.17 ; -1.0 ; ...}

MAIS PAS LE CONTRAIRE !

Qu'est-ce qu'un type ?

Attention

Une fois qu'un type de données est associé à une variable :

- cette variable ne peut plus en changer, et
- le contenu de cette variable doit obligatoirement être du même type.

Exemple

Si on définit : a : **entier** et b : **réel**

On peut :

- Donner à a les valeurs suivantes : {0 ; 2 ; -1 ; 1000 ; ...}
- Donner à b les valeurs suivantes : {0.5 ; 2.17 ; -1.0 ; ...}

MAIS PAS LE CONTRAIRE !

Qu'est-ce qu'un type ?

Les types les plus courants en Processing (et Java)

- **int** (ensemble $\mathbb{Z}$) : âge, nombre d'étages dans un immeuble, nombre d'habitants à Bordeaux...
- **float** (ensemble $\mathbb{R}$) : note à l'exam d'algo, aire d'un cercle...
- **char** : un caractère
- **boolean** (VRAI ou FAUX) : homme ? présent ? petit ?...

Qu'est-ce qu'un type ?

Les types les plus courants en Processing (et Java)

- **int** (ensemble $\mathbb{Z}$) : âge, nombre d'étages dans un immeuble, nombre d'habitants à Bordeaux...
- **float** (ensemble $\mathbb{R}$) : note à l'exam d'algo, aire d'un cercle...
- **char** : un caractère
- **boolean** (VRAI ou FAUX) : homme ? présent ? petit ?...

Qu'est-ce qu'un type ?

Les types les plus courants en Processing (et Java)

- **int** (ensemble $\mathbb{Z}$) : âge, nombre d'étages dans un immeuble, nombre d'habitants à Bordeaux...
- **float** (ensemble $\mathbb{R}$) : note à l'exam d'algo, aire d'un cercle...
- **char** : un caractère
- **boolean** (VRAI ou FAUX) : homme ? présent ? petit ?...

Qu'est-ce qu'un type ?

Les types les plus courants en Processing (et Java)

- **int** (ensemble $\mathbb{Z}$) : âge, nombre d'étages dans un immeuble, nombre d'habitants à Bordeaux...
- **float** (ensemble $\mathbb{R}$) : note à l'exam d'algo, aire d'un cercle...
- **char** : un caractère
- **boolean** (VRAI ou FAUX) : homme ? présent ? petit ?...

À vous de jouer !

```
1  int a = 3;
2  println("la variable a vaut ",a);
3  float b = -1.5;
4  println("la variable b vaut ",b);
5  char c = '@';
6  println("la variable c vaut ",c);
7  boolean d = true;
8  println("la variable d vaut ",d);
9  boolean e = false;
10 println("la variable e vaut ",e);
```

Affichage dans la fenêtre graphique :

```
1  text("a vaut "+a,10,10);
2  text("b vaut "+b,10,20);
3  text("c vaut "+c,10,30);
4  text("d vaut "+d,10,40);
5  text("e vaut "+e,10,50);
```

Notez, en Processing :

- Ctrl+T pour la mise en forme du code,
- clic droit / Rename pour renommer une variable dans tout le code.

Le mode Debug de Processing

Grâce à un debugger, on peut notamment :

- poser des points d'arrêt dans le programme
- consulter le contenu courant des variables à chaque arrêt
- exécuter le code d'un point d'arrêt à un autre, pas à pas, etc...

Un debugger est Indispensable lorsque :

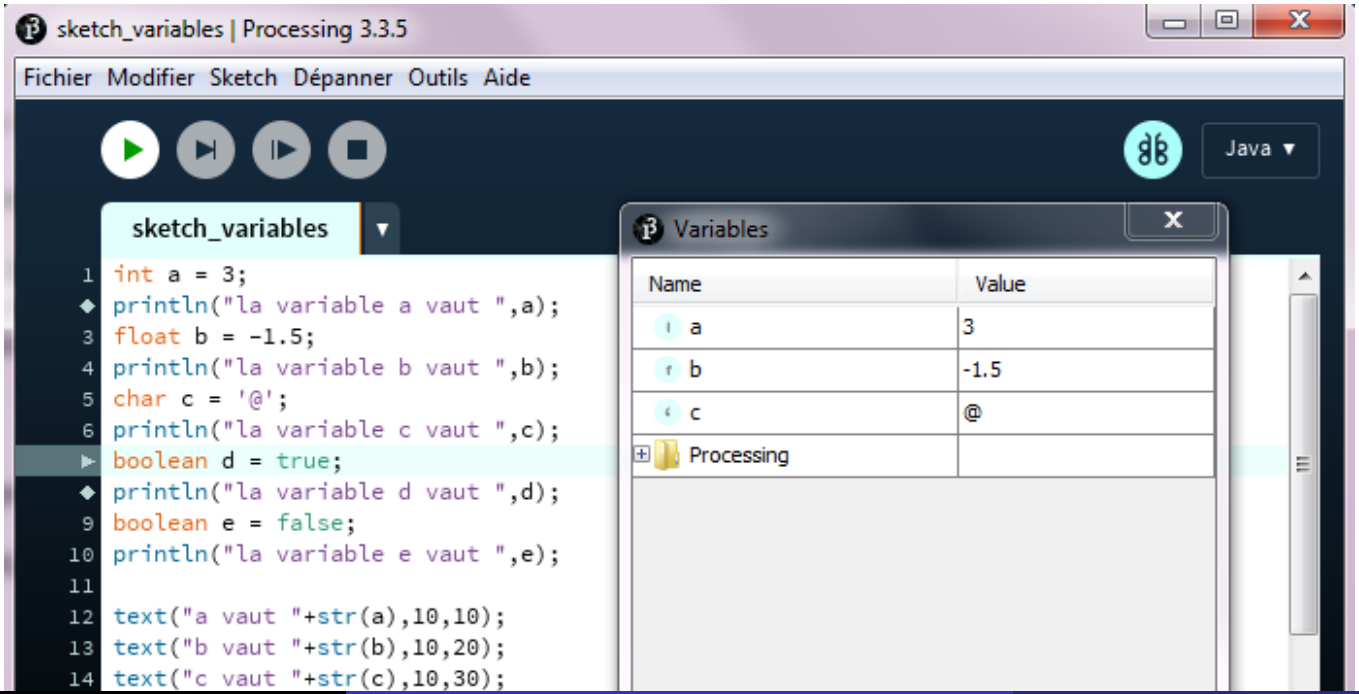
- le programme compile MAIS fournit un mauvais résultat ou plante à l'exécution

Un debugger est Utile aussi lorsque :

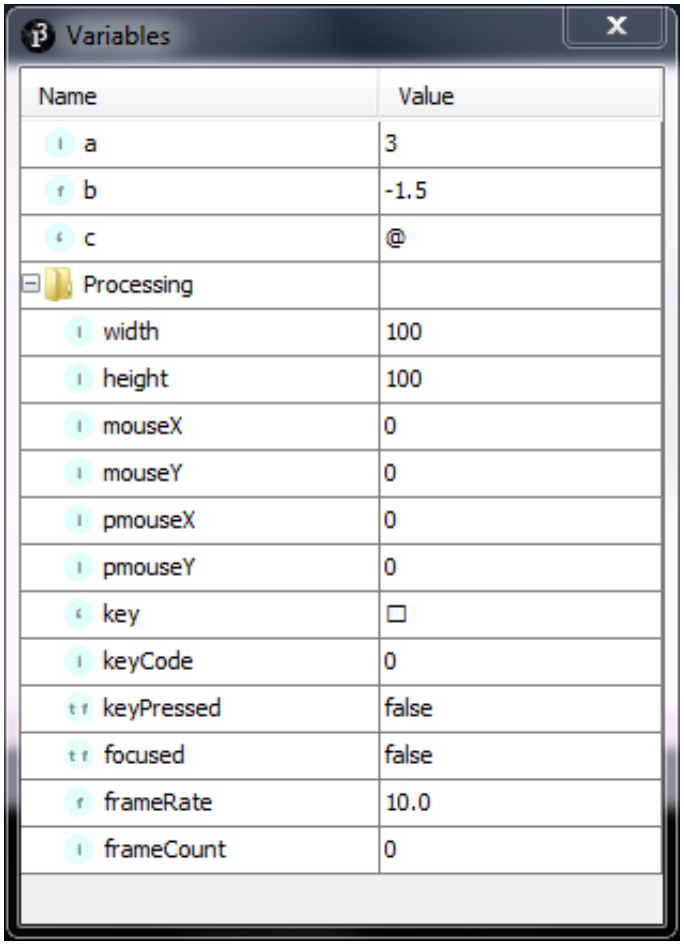
- on souhaite réaliser la "trace" du programme pour des vérifications en cours de développement

Le mode Debug de Processing

- Basculez en mode Debug en cliquant sur l'icône "coccinelle" en haut à droite
  - une fenêtre intitulée "Variables" s'ouvre
  - deux icônes apparaissent : Step (pas à pas) et Continuer
- Mettez un ou plusieurs points d'arrêt en cliquant sur le numéro de ligne (apparaît un losange), puis lancez l'exécution avec la flèche
- Consultez la valeur des variables dans la fenêtre "Variables" à chaque étape



Le mode Debug de Processing



- Les variables de base de Processing sont consultables
- Par exemple width et height peuvent être modifiées. Rajoutez en tout début de sketch :

```
1 size(400,500);
```

Les opérateurs sur les entiers, réels...

Que du classique :

- On retrouve tous les opérateurs connus : $/, +, -, *$
- Pour les entiers, l'opérateur / réalise la division entière : 11 / 2 vaut 5
- L'opérateur modulo % fournit le reste la division entière : 11 % 2 vaut 1
  Remarque : fonctionne aussi pour les réels (11.3 % 2 vaut 1.3), mais nous ne l'utiliserons pas.

Les opérateurs sur les entiers, réels...

Que du classique :

- On retrouve tous les opérateurs connus : $/, +, -, *$
- Pour les entiers, l'opérateur / réalise la division entière : 11 / 2 vaut 5
- L'opérateur modulo % fournit le reste la division entière : 11 % 2 vaut 1

  Remarque : fonctionne aussi pour les réels (11.3 % 2 vaut 1.3), mais nous ne l'utiliserons pas.

Les opérateurs sur les entiers, réels...

Que du classique :

- On retrouve tous les opérateurs connus : $/, +, -, *$
- Pour les entiers, l'opérateur / réalise la division entière : 11 / 2 vaut 5
- L'opérateur modulo % fournit le reste la division entière : 11 % 2 vaut 1
  Remarque : fonctionne aussi pour les réels (11.3 % 2 vaut 1.3), mais nous ne l'utiliserons pas.

A vous de jouer ! (entiers et réels)

```
1  int a = 11;
2  int b = 2;
3
4  println("a+b= ", a+b);
5  println("a-b= ", a-b);
6  println("a*b= ", a*b);
7  println("a/b= ", a/b);
8  println("a%b= ", a%b);
```

```
1  float a = 4.8;
2  float b = 2.0;
3
4  println("a+b= ", a+b);
5  println("a-b= ", a-b);
6  println("a*b= ", a*b);
7  println("a/b= ", a/b);
8  println("a%b= ", a%b);
```

A vous de jouer ! (caractères)

```
1  char a = '@';
2  char b = 'e';
3
4  println("a+b= ", a+b);
```

Remarque

L'opérateur + ne doit pas être utilisé sur les char, il a un comportement "inattendu"...

L'opérateur d'affectation

Exemple d'affectation

Soient a, b et c trois entiers et l'instruction d'affectation :
$c = a + b$

- On prend la valeur contenue dans la variable a
- On prend la valeur contenue dans la variable b
- On additionne ces deux valeurs
- On met ce résultat dans la variable c
- Si c possédait déjà une valeur, elle est alors perdue.

L'opérateur d'affectation

Exemple d'affectation

Soient a, b et c trois entiers et l'instruction d'affectation :
$c = a + b$

- On prend la valeur contenue dans la variable a
- On prend la valeur contenue dans la variable b
- On additionne ces deux valeurs
- On met ce résultat dans la variable c
- Si c possédait déjà une valeur, elle est alors perdue.

L'opérateur d'affectation

Exemple d'affectation

Soient a, b et c trois entiers et l'instruction d'affectation :
$c = a + b$

- On prend la valeur contenue dans la variable a
- On prend la valeur contenue dans la variable b
- On additionne ces deux valeurs
- On met ce résultat dans la variable c
- Si c possédait déjà une valeur, elle est alors perdue.

L'opérateur d'affectation

Exemple d'affectation

Soient a, b et c trois entiers et l'instruction d'affectation :
$c = a + b$

- On prend la valeur contenue dans la variable a
- On prend la valeur contenue dans la variable b
- On additionne ces deux valeurs
- On met ce résultat dans la variable c
- Si c possédait déjà une valeur, elle est alors perdue.

L'opérateur d'affectation

Exemple d'affectation

Soient a, b et c trois entiers et l'instruction d'affectation :
$c = a + b$

- On prend la valeur contenue dans la variable a
- On prend la valeur contenue dans la variable b
- On additionne ces deux valeurs
- On met ce résultat dans la variable c
- Si c possédait déjà une valeur, elle est alors perdue.

A vous de jouer !

Devinez l'affichage du programme suivant. Pour cela, effectuez sa trace en mode Debug.

```
1  int a, b, c;
2  a = 4;
3  b = 5;
4  c = a - b;
5  b = (a + b - c);
6  a = (a + b - c);
7  c = a + b;
8  println(a+b+c);
```

A vous de jouer ! (Méli-Mélo)

Les instructions ci-dessous sont mélangées. Il s'agit de les réordonner pour produire un programme Processing **correct**.

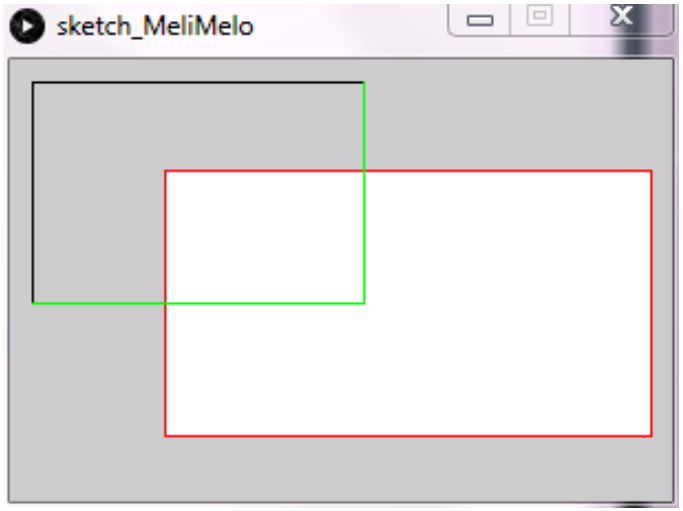
1. Combien de programmes Processing **corrects différents** peut-on produire ? Quels sont-ils ?
2. Effectuez manuellement la trace de chacun de ces programmes.
3. Vérifiez vos traces à l'aide du debugger.

```
1  y = 20;
2  rect(x,y,70,50);   // dessine un rectangle - consultez la documentation
3  int y;
4  y = y * 2;
5  int x = 10;
```

A vous de jouer ! (Méli-Mélo 2)

Les instructions ci-dessous sont mélangées. Il s'agit de les réordonner pour que le programme soit correct et produise le graphique ci-contre.

```
1  size(300,200);
2  strokeWeight(3);
3  rect(70,50,220,120);
4  stroke(255,0,0);
5  line(x1,y2,x2,y2);
6  int y2 = y1+100;
7  stroke(0,0,0);
8  line(x1,y1,x1,y2);
9  int y1 = 10;
10 int x2 = x1+150;
11 stroke(0,255,0);
12 int x1 = 10;
13 line(x2,y1,x2,y2);
14 line(x1,y1,x2,y1);
```



Les opérateurs booléens...

non

a	non a
Vrai	Faux
Faux	Vrai

A vous de jouer !

```
1  boolean a, b, resNon;
2  a = true;
3  b = false;
4
5  resNon = ! a;
6  println(resNon);
7
8  resNon = ! b;
9  println(resNon);
10
11 resNon = ! true;
12 println(resNon);
13
14 resNon = ! false;
15 println(resNon);
```

Les opérateurs booléens...

et

Au Restaurant Universitaire (RU), je peux :

- Manger et Dormir ⇒ FAUX
- Dormir et Manger ⇒ FAUX
- Manger et Discuter ⇒ VRAI
- Manger et Pratiquer le hand-ball ⇒ FAUX
- Manger et Discuter et Pratiquer le hand-ball ⇒ FAUX

a	**b**	**a et b**
Vrai	Vrai	Vrai
Faux	Faux	Faux
Vrai	Faux	Faux
Faux	Vrai	Faux

A vous de jouer !

```
1  boolean a, b, resEt;
2  a = true;
3  b = false;
4
5  resEt = a && b;
6  println(resEt);
7
8  resEt = b && a;
9  println(resEt);
10
11 resEt = a && a;
12 println(resEt);
```

Les opérateurs booléens...

ou (inclusif)

Au Restaurant Universitaire (RU), je peux :

- Manger ou Dormir ⇒ VRAI
- Dormir ou Manger ⇒ VRAI
- Manger ou Discuter ⇒ VRAI
- Dormir ou Pratiquer le hand-ball ⇒ FAUX
- Dormir ou Discuter ou Pratiquer le hand-ball ⇒ VRAI

a	b	a ou b
Vrai	Vrai	Vrai
Faux	Faux	Faux
Vrai	Faux	Vrai
Faux	Vrai	Vrai

A vous de jouer !

```
1  boolean a, b, resOu;
2  a = true;
3  b = false;
4
5  resOu = a || b;
6  println(resOu);
7
8  resOu = b || a;
9  println(resOu);
10
11 resOu = b || b;
12 println(resOu);
```

Les opérateurs de comparaison...

- L'opérateur d'égalité == permet d'évaluer si deux variables sont égales. Cette évaluation renvoie un booléen
- Sur le même modèle on dispose de l'opérateur d'inégalité : ! =
- Les autres opérateurs de comparaison : >, <, <=, >=

Attention

Ne pas confondre les opérateurs d'égalité == et d'affectation =

A vous de jouer !

Supprimez la variable booléenne `res` et remplacez-la par les variables de nom adéquat proposées en commentaire.

```
1  boolean res; // estDifferent, estEgal, estPlusPetit, estPair;
2  int a, b;
3  float c, d;
4  char e, f;
5  a = 2;
6  b = 6;
7  res = (a == b);
8  println(res);
9  a = 6;
10 b = 2;
11 res = (a%b == 0);
12 println(res);
13 c = -1.3;
14 d = 5.2;
15 res = (c <= d);
16 println(res);
17 e = '@';
18 f = '*';
19 res = (e != f);
20 println(res);
```

Opérateurs et types

Chacun de ces codes fonctionne-t-il ? Pourquoi ?

```
1  float reela = 2.4;
2  int entierb = 3;
3
4  entierb = reela;
5
6  println(reela," ",entierb);
```

```
1  float reela = 2.4;
2  int entierb = 3;
3
4  reela = entierb;
5
6  println(reela," ",entierb);
```

Le code ci-après ne produit pas ce qu'on attend...
Comment "forcer" le bon calcul ?

```
1  int valeur = 17;
2  float moitie = valeur / 2;
3
4  println(moitie);
```

Opérateurs et types : Réponses

Chacun de ces codes fonctionne-t-il ? Pourquoi ?

NON

```
1  float reela = 2.4;
2  int entierb = 3;
3
4  entierb = reela; // pb conversion
5
6  println(reela," ",entierb);
```

OUI

```
1  float reela = 2.4;
2  int entierb = 3;
3
4  reela = entierb;
5
6  println(reela," ",entierb);
```

Le code ci-après ne produit pas ce qu'on attend...
Comment "forcer" le bon calcul ?

```
1  int valeur = 17;
2  float moitie = (float)valeur / 2; // cast pour forcer division flottante
3
4  println(moitie);
```

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

A vous de jouer !

Mini-Projet 1 : Premières animations en Processing

Projet commencé en autonomie en séance.

- Introduction de setup() et draw()
- Gestion des événements souris

Pour la prochaine séance TP de la semaine

- Relire les premiers chapitres.
- Finir les exercices.
- Mini-Projet 1 : à terminer et remettre sur Moodle au plus tard en début de séance.

Mini-Projet 1 : Premières animations

Partie 1 : Rayons laser

1. Exécutez plusieurs fois ce programme qui colore le fond en noir, et affiche un rayon laser (ligne verte) aléatoire issu du centre.

```
1  background(0); // this makes the background black
2  stroke(0, 255, 0); // R, G, B
3  // the screen is 100 pixels wide and 100 pixels tall
4  // lines start at the middle of the screen (50, 50)
5  line(50, 50, random(100), random(100));
```

2. Entourez ce code avec `void draw(){...}` et testez.

```
1  void draw() {
2    background(0); // this makes the background black
3    stroke(0, 255, 0); // R, G, B
4    // the screen is 100 pixels wide and 100 pixels tall
5    // lines start at the middle of the screen (50, 50)
6    line(50, 50, random(100), random(100));
7  }
```

Mini-Projet 1 : Premières animations

Félicitations ! Vous venez de réaliser votre première animation !

draw()

La méthode draw() est exécutée automatiquement par le programme plusieurs fois par seconde : par défaut, 60 fois.

- Valeur par défaut modifiable en appelant frameRate(*fps*) (ex : $fps = 50$, 70, 1, etc...).
- Attention : frameRate() ne doit pas être utilisée dans draw(), mais dans setup() (cf juste après).

Mini-Projet 1 : Premières animations

setup()

La méthode setup() contient les instructions qui ne sont exécutées qu'UNE SEULE fois en tout début de programme.

Déplacez l'instruction background(0) dans le méthode setup(). Observez et expliquez la différence avec avant.

```
1  void setup() {
2    background(0); // this makes the background black
3  }
4  void draw() {
5    stroke(0, 255, 0); // R, G, B
6    // the screen is 100 pixels wide and 100 pixels tall
7    // lines start at the middle of the screen (50, 50)
8    line(50, 50, random(100), random(100));
9  }
```

Mini-Projet 1 : Premières animations

Vitesse des rayons

Essayez frameRate(*fps*), comme indiqué précédemment, pour régler à votre guise la vitesse des rayons.

Effet lumineux

Pour un meilleur effet lumineux des rayons, rendez aléatoire la composante verte d'affichage : random(255) au lieu de 255.

Origine aléatoire des rayons

Les rayons sont pour l'instant issus du centre (50,50).
Modifiez le code pour que, à chaque exécution, les rayons soient issus d'une seule et même origine aléatoire.
Indication : deux variables sont nécessaires.

Mini-Projet 1 : Premières animations

Partie 2 : Vaisseau furtif

On souhaite qu'un vaisseau, représenté par une ellipse, se déplace tout seul à l'écran (trajectoire horizontale vers la droite).
On part du code à compléter ci-dessous.

```
1  int vaisseauX;
2  void setup() {
3    size(300, 200);
4    noStroke();
5    vaisseauX = 30;
6  }
7  void draw() {
8    ellipse(vaisseauX, 100, 50, 15);
9  //  vaisseauX = ?? ;
10 }
```

1. Pourquoi l'instruction size() ligne 3 est-elle dans setup() et non dans draw() ? Que fait noStroke() ligne 4 ?
2. Complétez la ligne 9 pour que le vaisseau s'anime.
3. Est-ce suffisant ? Que rajouter pour éviter la "trainée" ?

Mini-Projet 1 : Premières animations

Partie 3 : Gestion de la souris

Vaisseau furtif : arrêt

Objectif : le vaisseau arrête de bouger lorsqu'un bouton de la souris est pressé.

Reprenez le code précédent et rajoutez :

1. (si ce n'est pas déjà fait) une variable vitesse qui contient le nombre de pixels dont se déplace le vaisseau à chaque étape
2. la méthode appelée lorsqu'un bouton de la souris est pressé

```
1 void mousePressed() {
2   vitesse = 0;
3 }
```

Mini-Projet 1 : Premières animations

Vaisseau furtif : changement de direction

1. Au lieu de s'arrêter lorsqu'un bouton de la souris est pressé, le vaisseau change de direction (gauche/droite).
2. Le vaisseau s'arrête lorsqu'on détecte un cliqué-glissé (mouseDragged).

Vaisseau furtif : hyper-espace (jump)

1. Au lieu de s'arrêter lorsqu'on détecte un cliqué-glissé (mouseDragged), le vaisseau apparaît instantanément aux coordonnées de la souris.

Dépendance des méthodes mouseXXX() avec draw()

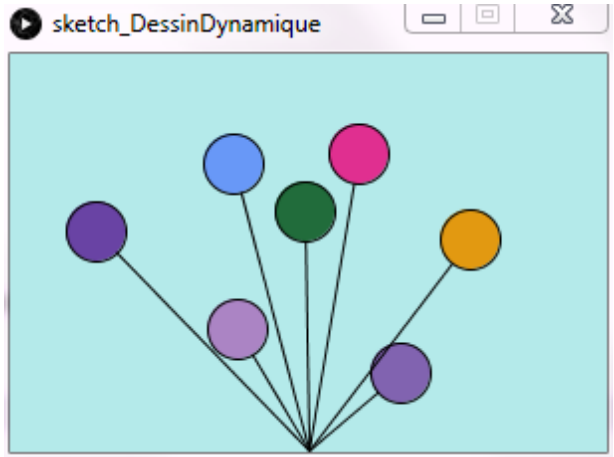
Les méthodes mousePressed(), etc ... ne sont actives que si le programme contient une méthode draw(), même si elle est vide.

Mini-Projet 1 : Premières animations

Partie 4 : Ballons multicolores

Stand de ballons : à chaque clic de souris, un ballon est dessiné à l'endroit du clic et relié par un fil au stand.
Réorganisez les instructions, avec les méthodes setup(), draw() et mousePressed().

```
1  int origineX;
2  line(origineX, origineY, mouseX, mouseY);
3  size(300, 200);
4  ellipse(mouseX, mouseY, 30, 30);
5  origineX = width/2;
6  fill(random(255), random(255), random(255));
7  origineY = height;
8  background(180, 234, 234);
9  int origineY;
```



Remarquez le rôle des variables mouseX, mouseY, width et height de Processing.

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

Avant de passer à la suite...

Correction du Mini-Projet 1 : Premières animations

L'enseignant commente une correction en vidéo-projection.

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

Les tableaux

Exemple

Imaginons que l'on veuille afficher un ciel étoilé de 5 étoiles.
Une solution possible :

```
1  float x1,x2,x3,x4,x5; //float au lieu de int pour eviter cast des random()
2  float y1,y2,y3,y4,y5;
3  x1 = random(width); y1 = random(height);
4  x2 = random(width); y2 = random(height);
5  x3 = random(width); y3 = random(height);
6  x4 = random(width); y4 = random(height);
7  x5 = random(width); y5 = random(height);
8
9  int maxT = 5;
10 background(0);
11 stroke(255);
12 strokeWeight(random(maxT)); point(x1, y1);
13 strokeWeight(random(maxT)); point(x2, y2);
14 strokeWeight(random(maxT)); point(x3, y3);
15 strokeWeight(random(maxT)); point(x4, y4);
16 strokeWeight(random(maxT)); point(x5, y5);
```

Les tableaux

Imaginons que l'on veuille 1000 étoiles.

Exemple

- Il faut déclarer 1000 variables x et 1000 variables y
- Il faut les initialiser, 1000 fois :
  xi = random(width) ; yi = random(height) ;
- Il faut les dessiner, 1000 fois :
  strokeWeight(random(maxT)) ; point(xi, yi) ;

Les tableaux

Il est possible de rassembler toutes ces variables en une seule, au sein de laquelle chaque valeur sera désignée par un numéro.

Déclaration

type[] *nomDuTableau* ;

Initialisation

***nomDuTableau* = new type[*taille*] ;**

Exemples de tableaux

```
1  // Declaration puis initialisation d'un tableau de 5 entiers
2  int[] monTab;
3  monTab = new int[5];
4
5  // Possible : declaration et initialisation en 1 seule instruction
6  char[] tab2 = new char[6]; //tableau de 6 caracteres
7  float[] ceJoliTableau = new float[1000]; //tableau de 1000 reels
```

A vous de jouer !

Déclarez et initialisez

- un tableau de 3 entiers
- un tableau de 1 réel
- un tableau de 10 booléens
- un tableau de 4 caractères

Les tableaux

Accès - Affectation

Un tableau de n cases indice ses cases de 0 à $n-1$.
L'affectation d'une case se fait simplement avec l'opérateur =.

Exemple de tableau

```
1  // Declaration et initialisation du tableau
2  int[] monTab = new int[4];
3  // Initialisation du contenu de chaque case du tableau
4  monTab[0] = 0;
5  monTab[1] = 3;
6  monTab[2] = -1;
7  monTab[3] = 4333;
```

monTab :

monTab[0]	*monTab*[1]	*monTab*[2]	*monTab*[3]
0	3	-1	4333

Les tableaux

Exemple de tableau

```
1  // Declaration et initialisation du tableau
2  char[] tab2 = new char[6];
3  // Initialisation du contenu de chaque case du tableau
4  tab2[0] = 'a';
5  tab2[1] = '4';
6  tab2[2] = 'd';
7  tab2[3] = '@';
8  tab2[4] = 'e';
9  tab2[5] = 'a';
```

tab2 :

tab2[0]	tab2[1]	tab2[2]	tab2[3]	tab2[4]	tab2[5]
'a'	'4'	'd'	'@'	'e'	'a'

Les tableaux

ATTENTION

- En Processing, le premier indice d'un tableau est toujours 0
- Tous les éléments d'un tableau sont du même type
- Un tableau est de taille fixe et pré-définie
- Il faut toujours penser à initialiser
  - la taille d'un tableau
  - et le contenu de chaque case d'un tableau

Les tableaux

ATTENTION

- En Processing, le premier indice d'un tableau est toujours 0
- Tous les éléments d'un tableau sont du même type
- Un tableau est de taille fixe et pré-définie
- Il faut toujours penser à initialiser
  - la taille d'un tableau
  - et le contenu de chaque case d'un tableau

Les tableaux

ATTENTION

- En Processing, le premier indice d'un tableau est toujours 0
- Tous les éléments d'un tableau sont du même type
- Un tableau est de taille fixe et pré-définie
- Il faut toujours penser à initialiser
  - la taille d'un tableau
  - et le contenu de chaque case d'un tableau

Les tableaux

ATTENTION

- En Processing, le premier indice d'un tableau est toujours 0
- Tous les éléments d'un tableau sont du même type
- Un tableau est de taille fixe et pré-définie
- Il faut toujours penser à initialiser
  - la taille d'un tableau
  - et le contenu de chaque case d'un tableau

Les tableaux

ATTENTION

- En Processing, le premier indice d'un tableau est toujours 0
- Tous les éléments d'un tableau sont du même type
- Un tableau est de taille fixe et pré-définie
- Il faut toujours penser à initialiser
  - la taille d'un tableau
  - et le contenu de chaque case d'un tableau

Les tableaux

ATTENTION

- En Processing, le premier indice d'un tableau est toujours 0
- Tous les éléments d'un tableau sont du même type
- Un tableau est de taille fixe et pré-définie
- Il faut toujours penser à initialiser
  - la taille d'un tableau
  - et le contenu de chaque case d'un tableau

Trace

```
int a,b;
int[] tab = new int[3];
a=5;
b=2+a;
tab[0]=a+b;
tab[1]=tab[0]-b;
tab[2]=tab[1]-tab[0];
tab[0]=tab[0]-tab[2];
```

a	b	tab		
		tab[0]	tab[1]	tab[2]
5	?	?	?	?
5	7	?	?	?
5	7	12	?	?
5	7	12	5	?
5	7	12	5	-7
5	7	19	5	-7

Trace

```
int a,b;
int[] tab = new int[3];
a=5;
b=2+a;
tab[0]=a+b;
tab[1]=tab[0]-b;
tab[2]=tab[1]-tab[0];
tab[0]=tab[0]-tab[2];
```

a	b	tab		
		tab[0]	tab[1]	tab[2]
5	?	?	?	?
5	7	?	?	?
5	7	12	?	?
5	7	12	5	?
5	7	12	5	-7
5	7	19	5	-7

Trace

```
int a,b;
int[] tab = new int[3];
a=5;
b=2+a;
tab[0]=a+b;
tab[1]=tab[0]-b;
tab[2]=tab[1]-tab[0];
tab[0]=tab[0]-tab[2];
```

a	b	tab		
		tab[0]	tab[1]	tab[2]
5	?	?	?	?
5	7	?	?	?
5	7	12	?	?
5	7	12	5	?
5	7	12	5	-7
5	7	19	5	-7

Trace

```
int a,b ;
int[] tab = new int[3] ;
a=5 ;
b=2+a ;
tab[0]=a+b ;
tab[1]=tab[0]-b ;
tab[2]=tab[1]-tab[0] ;
tab[0]=tab[0]-tab[2] ;
```

a	b	tab		
		tab[0]	tab[1]	tab[2]
5	?	?	?	?
5	7	?	?	?
5	7	12	?	?
5	7	12	5	?
5	7	12	5	-7
5	7	19	5	-7

Trace

```
int a,b ;
int[] tab = new int[3] ;
a=5 ;
b=2+a ;
tab[0]=a+b ;
tab[1]=tab[0]-b ;
tab[2]=tab[1]-tab[0] ;
tab[0]=tab[0]-tab[2] ;
```

a	b	tab		
		tab[0]	tab[1]	tab[2]
5	?	?	?	?
5	7	?	?	?
5	7	12	?	?
5	7	12	5	?
5	7	12	5	-7
5	7	19	5	-7

Trace

```
int a,b ;
int[] tab = new int[3] ;
a=5 ;
b=2+a ;
tab[0]=a+b ;
tab[1]=tab[0]-b ;
tab[2]=tab[1]-tab[0] ;
tab[0]=tab[0]-tab[2] ;
```

a	b	tab		
		tab[0]	tab[1]	tab[2]
5	?	?	?	?
5	7	?	?	?
5	7	12	?	?
5	7	12	5	?
5	7	12	5	-7
5	7	19	5	-7

A vous de jouer !

Dans chaque cas, déclarez et choisissez le type d'une variable qui vous semble le plus approprié :

- Note d'un étudiant à un examen de math
- Numéro de téléphone d'une personne
- Adresse d'une personne
- Le fait qu'une personne soit un homme ou pas
- Notes des 100 étudiants de la promo en math

A vous de jouer !

- Déclarez et initialisez un tableau de 5 entiers et un tableau de 10 caractères
- Écrivez, compilez et exécutez le code source permettant d'afficher la première et la dernière case de ces deux tableaux. Qu'observez-vous ?
- Écrivez le code source permettant d'affecter à la première et dernière case de ces deux tableaux une valeur que vous choisirez et ensuite affichez-les.

Remarque importante

Une variable doit toujours être initialisée, même si c'est à zéro, avant toute utilisation de son contenu pour un calcul, un affichage, etc.

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

Les conditionnelles

Dans les exemples donnés jusqu'ici, nous avons appliqué séquentiellement les algorithmes.
Il se peut que l'on doive conditionner l'exécution de certaines instructions.

Intérêt d'une instruction conditionnelle

- Ne pas exécuter une conversion de dollars en euros si la somme est négative
- Ne pas calculer la division de 2 nombres si le dénominateur $= 0$

Les conditionnelles

Dans les exemples donnés jusqu'ici, nous avons appliqué séquentiellement les algorithmes.
Il se peut que l'on doive conditionner l'exécution de certaines instructions.

Intérêt d'une instruction conditionnelle

- Ne pas exécuter une conversion de dollars en euros si la somme est négative
- Ne pas calculer la division de 2 nombres si le dénominateur $= 0$

L'instruction if

Syntaxe

if *(Expression booléenne)*
{
    Instructions réalisées si l'expression a été évaluée à VRAI
}
else
{
    Instructions réalisées si l'expression a été évaluée à FAUX
}

L'instruction if

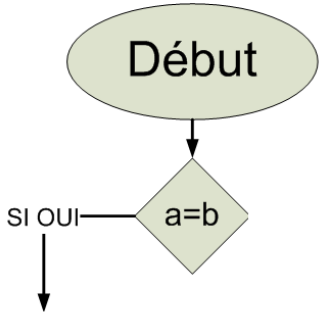
```
1  float note = 16.5;
2
3  if (note>=10)
4  {
5    println("Vous avez la moyenne");
6  }
7
8  if ((note<10) && (note>=8))
9  {
10   println("note non eliminatoire");
11 }
12
13 if (note<8)
14 {
15   println("note eliminatoire");
16 }
```

L'instruction if

```
1  float note = 16.5;
2
3  if (note>=10)
4  {
5    println("Vous avez la moyenne");
6  }
7  else
8  {
9    if (note>=8)   // remarquez que le test (note<10) n'est plus utile
10   {
11     println("note non eliminatoire");
12   }
13   else  // signifie ici que note<8
14   {
15     println("note eliminatoire");
16   }
17 }
```

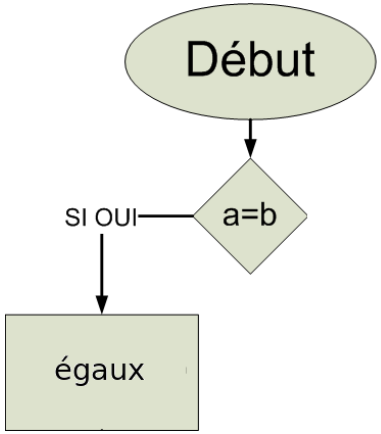
Exemple

```
int a,b;
a=2;
b=2;
if (a==b)
{
    println("égaux");
}
else
{
    if (a<b)
    {
        println("a plus petit que b");
    }
    else
    {
        println("b plus petit que a");
    }
}
```



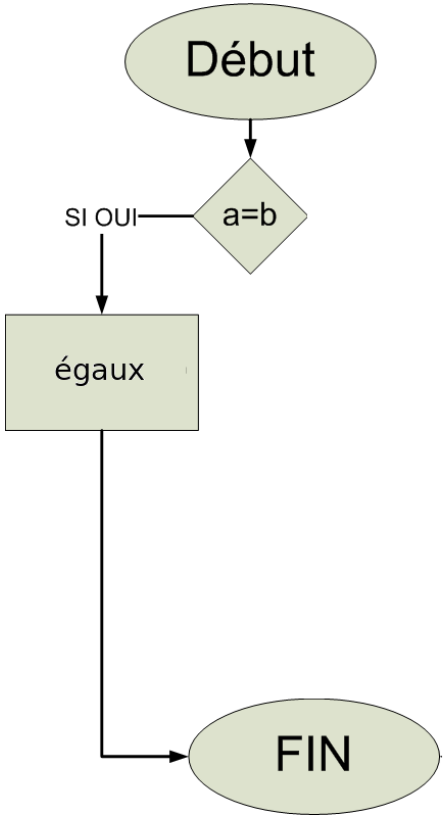
Exemple

```
int a,b ;
a=2 ;
b=2 ;
if (a==b)
{
    println("égaux") ;
}
else
{
    if (a<b)
    {
        println("a plus petit que b") ;
    }
    else
    {
        println("b plus petit que a") ;
    }
}
```



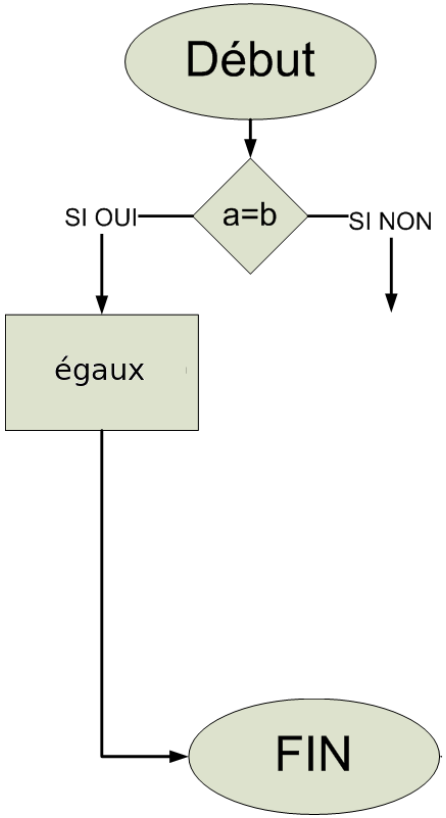
Exemple

```
int a,b ;
a=2 ;
b=2 ;
if (a==b)
{
    println("égaux") ;
}
else
{
    if (a<b)
    {
        println("a plus petit que b") ;
    }
    else
    {
        println("b plus petit que a") ;
    }
}
```



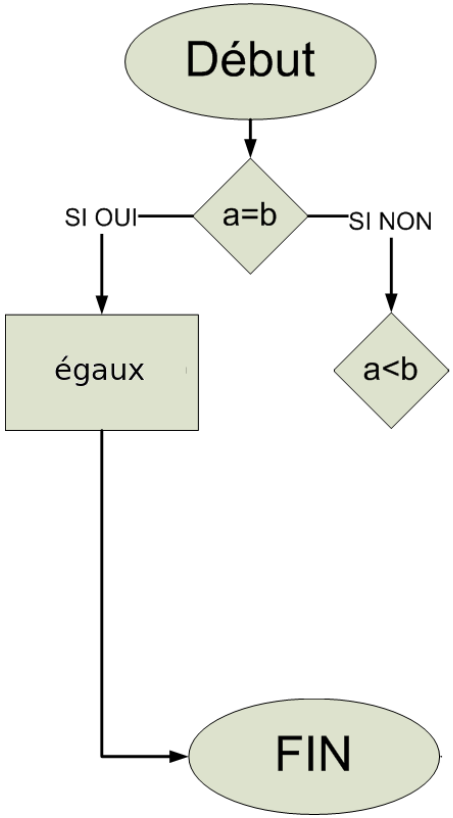
Exemple

```
int a,b;
a=2;
b=2;
if (a==b)
{
    println("égaux");
}
else
{
    if (a<b)
    {
        println("a plus petit que b");
    }
    else
    {
        println("b plus petit que a");
    }
}
```



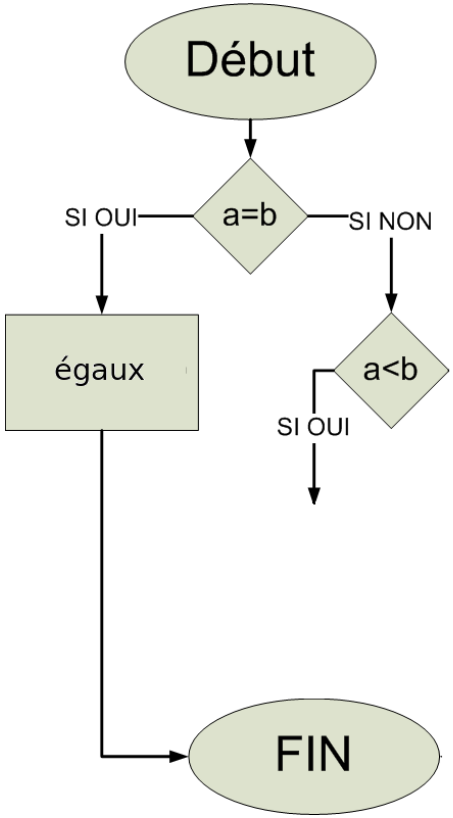
Exemple

```
int a,b;
a=2;
b=2;
if (a==b)
{
    println("égaux");
}
else
{
    if (a<b)
    {
        println("a plus petit que b");
    }
    else
    {
        println("b plus petit que a");
    }
}
```



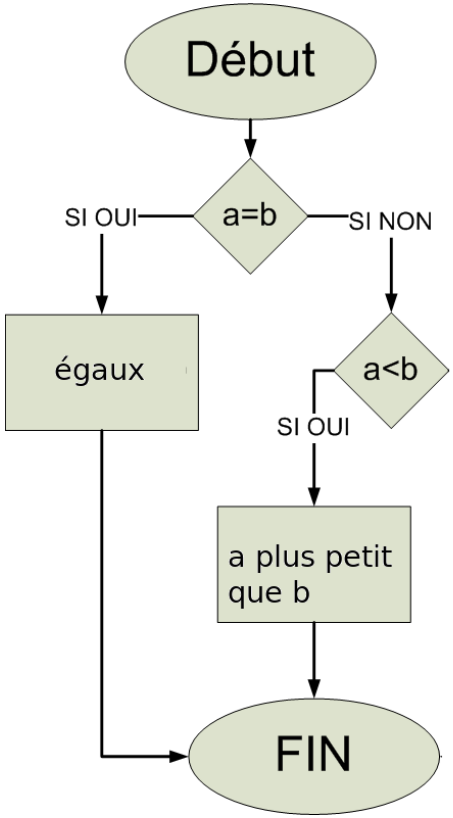
Exemple

```
int a,b;
a=2;
b=2;
if (a==b)
{
    println("égaux");
}
else
{
    if (a<b)
    {
        println("a plus petit que b");
    }
    else
    {
        println("b plus petit que a");
    }
}
```



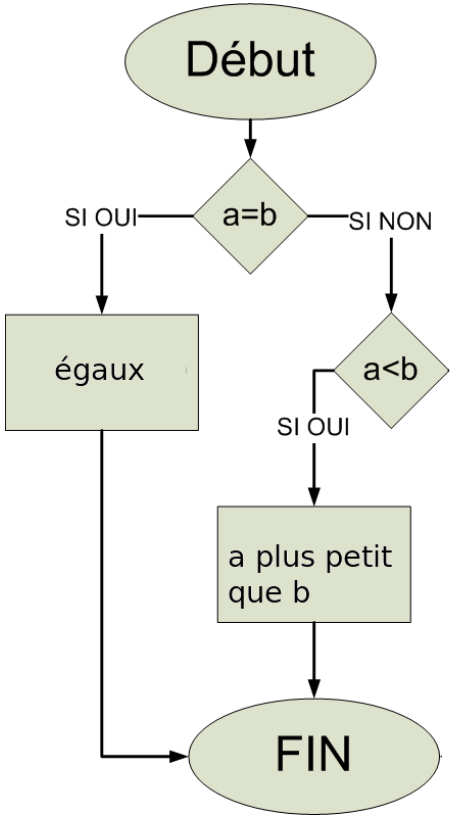
Exemple

```
int a,b;
a=2;
b=2;
if (a==b)
{
    println("égaux");
}
else
{
    if (a<b)
    {
        println("a plus petit que b");
    }
    else
    {
        println("b plus petit que a");
    }
}
```



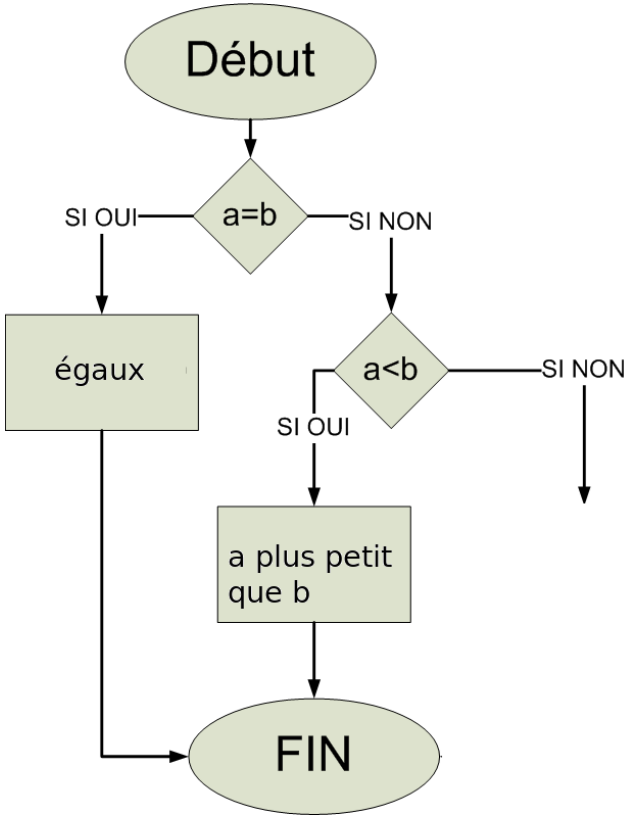
Exemple

```
int a,b;
a=2;
b=2;
if (a==b)
{
    println("égaux");
}
else
{
    if (a<b)
    {
        println("a plus petit que b");
    }
    else
    {
        println("b plus petit que a");
    }
}
```



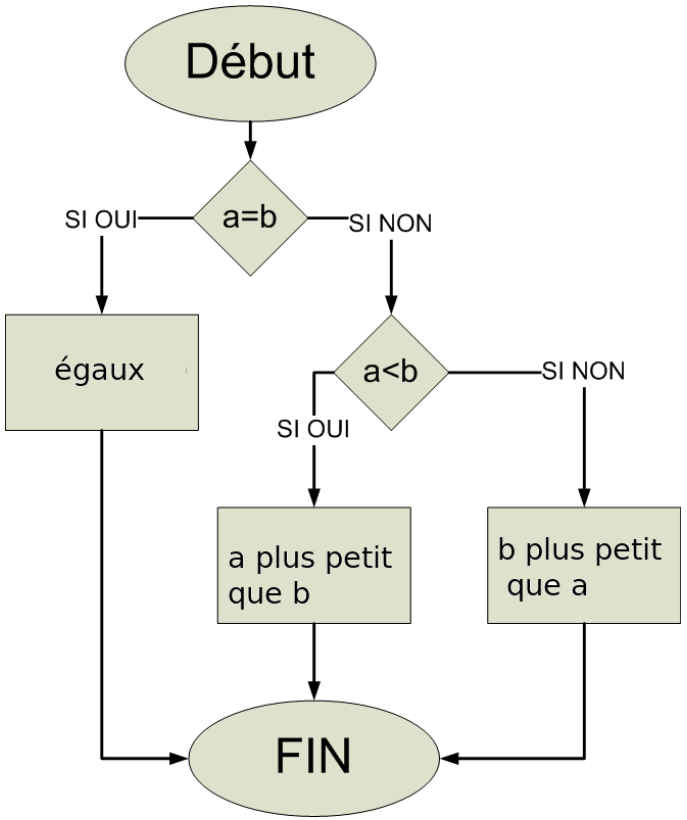
Exemple

```
int a,b;
a=2;
b=2;
if (a==b)
{
    println("égaux");
}
else
{
    if (a<b)
    {
        println("a plus petit que b");
    }
    else
    {
        println("b plus petit que a");
    }
}
```



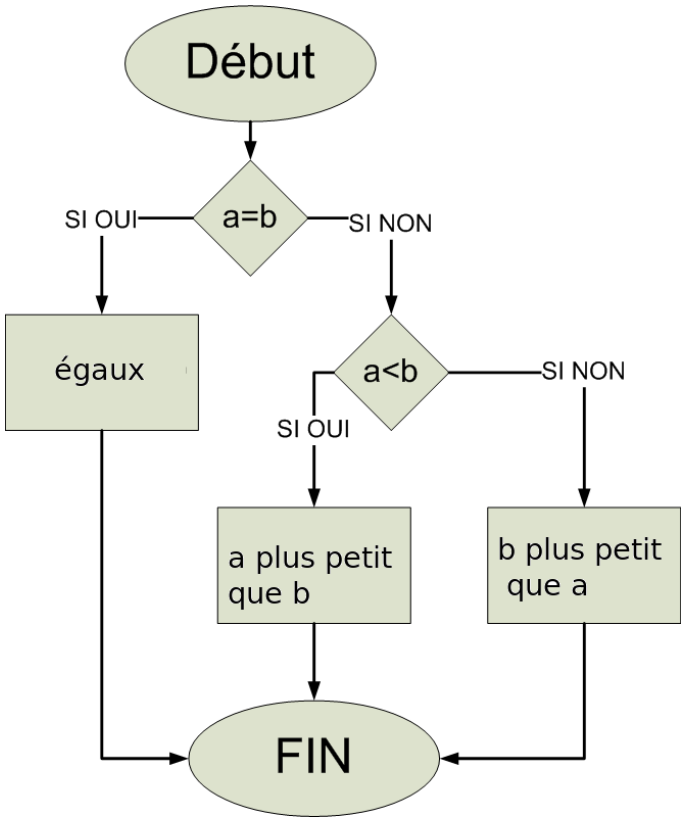
Exemple

```
int a,b ;
a=2 ;
b=2 ;
if (a==b)
{
    println("égaux") ;
}
else
{
    if (a<b)
    {
        println("a plus petit que b") ;
    }
    else
    {
        println("b plus petit que a") ;
    }
}
```



Exemple

```
int a,b ;
a=2 ;
b=2 ;
if (a==b)
{
    println("égaux") ;
}
else
{
    if (a<b)
    {
        println("a plus petit que b") ;
    }
    else
    {
        println("b plus petit que a") ;
    }
}
```



Remarque : blocs avec 1 seule instruction

```
int a,b ;
a=2 ;
b=2 ;
if (a==b)
{
    println("égaux") ;
}
else
{
    if (a<b)
    {
        println("a plus petit que b") ;
    }
    else
    {
        println("b plus petit que a") ;
    }
}
```

Chaque bloc réduit à 1 seule instruction peut s'écrire sans les accolades.

```
int a,b ;
a=2 ;
b=2 ;
if (a==b)
    println("égaux") ;
else if (a<b)
    println("a plus petit que b") ;
else
    println("b plus petit que a") ;
```

A vous de jouer !

Attention !

Parfois, ne pas écrire les accolades est **opportun**, comme ici...

```
1  void draw() {
2  }
3
4  void keyPressed() {
5    if (key=='a')
6      println("Voyelle");
7    else if (key=='e')
8      println("Voyelle");
9    else if (key=='i')
10     println("Voyelle");
11   else if (key=='o')
12     println("Voyelle");
13   else if (key=='u')
14     println("Voyelle");
15   else if (key=='y')
16     println("Voyelle");
17   else
18     println("Consonne");
19 }
```

```
1  // Indentation plus visuelle
2
3
4  void keyPressed() {
5    if (key=='a')
6      println("Voyelle");
7    else if (key=='e')
8            println("Voyelle");
9          else if (key=='i')
10                 println("Voyelle");
11               else if (key=='o')
12                     println("Voyelle");
13                   else if (key=='u')
14                         println("Voyelle");
15                       else if (key=='y')
16                             println("Voyelle");
17                           else
18                             println("Consonne");
19 }
```

A vous de jouer !

Attention !

MAIS le plus souvent, ne pas écrire les accolades est source **d'erreur**, comme ici... Expliquez les affichages obtenus.

```
1  int a = 4;
2  char b = 'a';
3  char c = 'a';
4  int d = 4;
5  if (a==d)
6    println("a et d sont egaux");
7  if (a<=d)
8    println("a est plus petit ou egal a d");
9  if (a==d)
10   println("a et d sont egaux");
11 else if (a<=d)
12   println("a est plus petit ou egal a d");
13 if (b==c)
14 {
15   println("b vaut ", b, " et c vaut ", c);
16   println("donc b et c sont egaux");
17 }
18 else
19   println("b vaut ", b," et c vaut ", c);
20   println("donc b et c sont differents");
```

Conseil concernant les accolades

Conseil (voire Obligation !)

- Utilisez **systématiquement** les accolades pour délimiter vos blocs "if" "else".
- Seule exception éventuellement :
  dans une suite de "if" "else if" avec des blocs de 1 instruction comme l'exercice précédent sur les voyelles et les consonnes.

A vous de jouer !

Expliquez pourquoi la compilation de ce programme provoque un message d'erreur. Corrigez cette erreur (qui est classique !)

```
1  int a = 4;
2  int b = 5;
3
4  if (a=b)
5  {
6     println("a et b sont egaux");
7  }
8  else
9  {
10    println("a et b sont differents");
11 }
```

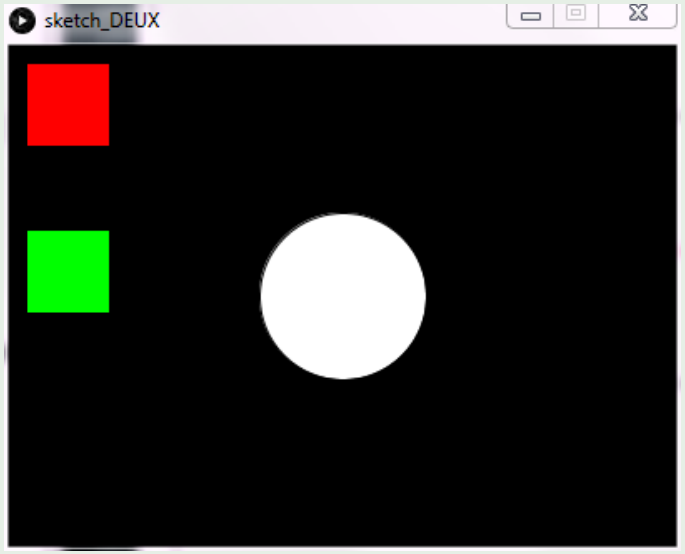
A vous de jouer !

Sur papier, effectuez la trace de l'algorithme suivant. Ensuite, écrivez le code, compilez, exécutez et vérifiez votre réponse.

```
1  int[] tab = new int[3];
2  boolean trouve;
3  int nb = -4;
4  tab[0]=4; tab[1]=-4; tab[2]=10;
5
6  if (tab[0]==nb)
7    trouve = true;
8  else if (tab[1]==nb)
9    trouve = true;
10 else if (tab[2]==nb)
11   trouve = true;
12 else
13   trouve = false;
14
15 if (!trouve)
16 {
17   println("Nombre", nb, "non trouve");
18 }
19 else
20 {
21    println("Nombre", nb, "trouve");
22 }
```

A vous de jouer !

Zones de clic



- Si on clique sur un carré de couleur, le rond prend cette couleur.
- Si on clique n'importe où ailleurs, le rond prend sa couleur blanche d'origine.

A vous de jouer !

Voici le programme, mais il est buggé...

1. Effectuez la trace en mode Debug pour le corriger.
2. Rajoutez un carré de couleur bleu et sa gestion.

```
color rouge = color(255,0,0);
color vert = color(0,255,0);
color blanc = color(255,255,255);
color valeur;

void setup() {
  size(400,300);
  valeur = blanc;
}

void draw() {
  background(0);
  fill(rouge); rect(10, 10,50,50);
  fill(vert);  rect(10,110,50,50);
  fill(valeur);
  ellipse(width/2,height/2,100,100);
}
```

```
void mousePressed() {
  // si dans carre rouge
  if ((mouseX>=10 && mouseX<=60)
   || (mouseY>=10 && mouseY<=60))
  {
    valeur = rouge;
  }
  // sinon si dans carre vert
  else if ((mouseX>=10 && mouseX<=60)
        || (mouseY>=110 && mouseY<=160))
  {
    valeur = vert;
  }
  // sinon n'importe ou ailleurs
  else
  {
    valeur = blanc;
  }
}
```

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

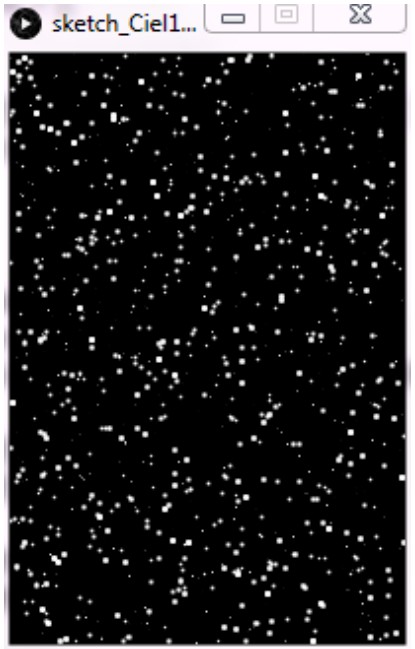
Quand le séquentiel ne suffit plus

Chercher le nombre 52 dans un tableau et afficher le mot "TROUVE !" si c'est effectivement le cas.

```
1  int[] tab = new int[100];
2
3  tab[0] = -4;
4  tab[1] = 342;
5  ...
6  ...
7  tab[98] = 52;
8  tab[99] = 3;
9
10 if (tab[0]==52)
11   println("TROUVE !");
12 else if (tab[1]==52)
13   println("TROUVE !");
14 ...
15 ...
16 else if (tab[99]==52)
17   println("TROUVE !");
```

Quand le séquentiel ne suffit plus

Générer et afficher à l'écran un ciel étoilé de 1000 étoiles.




```
1  size(200,300);
2  float[] x = new float[1000];
3  float[] y = new float[1000];
4
5  x[0] = random(width); y[0] = random(height);
6  x[1] = random(width); y[1] = random(height);
7  ...
8  ...
9  x[999] = random(width); y[999] = random(height);
10
11 int maxT = 3;
12 background(0);
13 stroke(255);
14
15 strokeWeight(random(maxT)); point(x[0], y[0]);
16 strokeWeight(random(maxT)); point(x[1], y[1]);
17 ...
18 ...
19 strokeWeight(random(maxT)); point(x[999], y[999]);
```

Quand le séquentiel ne suffit plus

Remarque

D'un point de vue algorithme il n'y a pas d'erreur, mais c'est "quasi" impossible à mettre réellement en place. Heureusement, il existe des commandes de répétition !

Définition

Il existe deux types d'itérations :

1. Les déterministes : le nombre de boucles est défini à l'entrée de la boucle.
2. Les indéterministes : l'exécution de la prochaine boucle est conditionnée par une expression booléenne.

La boucle POUR

Pour afficher 10 fois "bonjour"

Jusqu'ici, on sait faire :

```
println("bonjour");
/*...*/
println("bonjour");
```

On aimerait plutôt faire :

```
Faire 10 fois :
{
   println("bonjour");
}
```

Pour afficher les nombres de 1 à 10

Jusqu'ici, on sait faire :

```
println(1);
/*...*/
println(10);
```

On aimerait plutôt faire :

```
Pour toutes les valeurs de i entre 1 et 10 :
{
   println(i);
}
```

Nous voulons donc parcourir un intervalle (ici, de 1 à 10).

La boucle POUR

Nous voulons donc parcourir un intervalle (ici, de 1 à 10).

La boucle POUR sert à ça !

Il faut donc préciser :	dans notre exemple :
un point de départ	i = 1
une condition pour entrer dans la boucle	i <= 10
que faire pour obtenir la valeur suivante	i++

Ainsi, en Java, la boucle POUR se "pense" comme ça :

```
i = 1;
faire le code en dessous tant que i <= 10
    // traitement, par exemple : println(i);
    i++;
```

mais "s'écrit" de manière plus compacte...

La boucle POUR

Idée

```
for (point de départ ;
     condition pour rentrer dans la boucle ;
     changement à faire à la fin de chaque tour)
{
     Instructions à réaliser dans la boucle
}
```

Syntaxe

```
for (int i = 1 ; i <= 10 ; i++)
{
     Instructions à réaliser dans la boucle
}
```

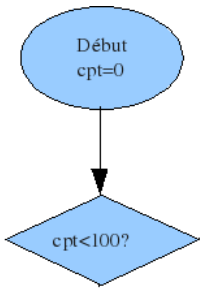
La boucle POUR

```
1  for (int i=0; i<=2; i++) {
2    println(i);
3  }
4  print("sortie de boucle : ");
5  println(i);
```

1. i vaut 0
2. i est $\leq$ à 2
3. on affiche i qui vaut 0
4. i vaut maintenant 1,
5. i est $\leq$ à 2
6. on affiche i qui vaut 1
7. i vaut maintenant 2,
8. i est $\leq$ à 2
9. on affiche i qui vaut 2
10. i vaut maintenant 3,
11. i est n'est pas $\leq$ à 2. On sort de la boucle.
12. Affiche "sortie de boucle : 3".

Exemple : Recherche dans un tableau

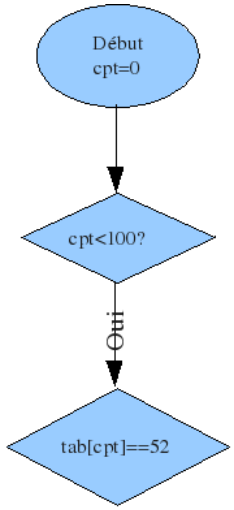
```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```



cpt	**0**	**1**	**2**	**3**	**4**	**...**	**99**
tab[cpt]	20	52	-1	52	0	...	45

Exemple : Recherche dans un tableau

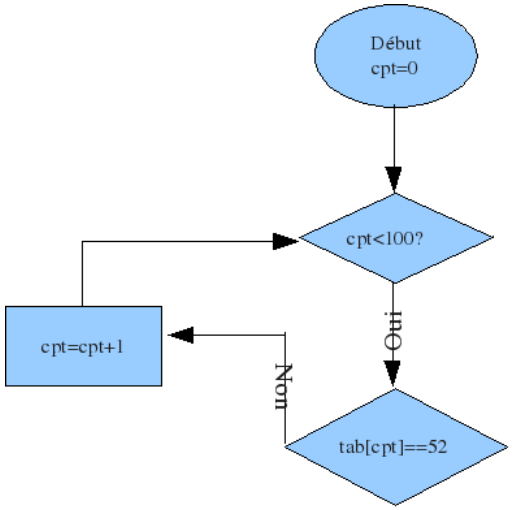
```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```



cpt	0	1	2	3	4	...	99
tab[cpt]	20	52	-1	52	0	...	45

Exemple : Recherche dans un tableau

```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```

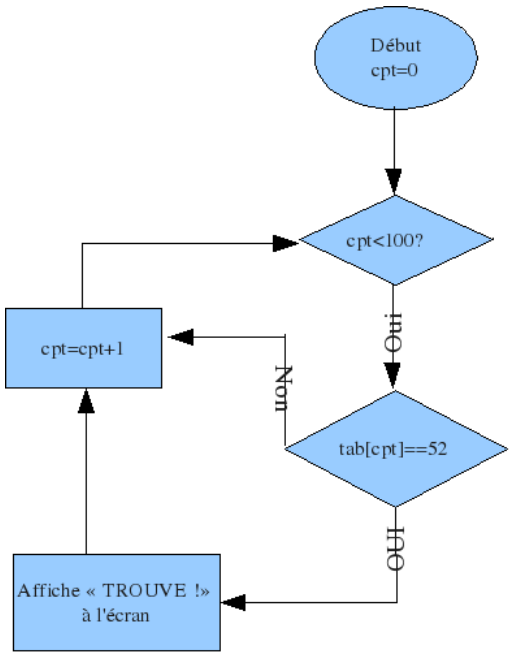


cpt	0	1	2	3	4	...	99
tab[cpt]	20	52	-1	52	0	...	45

Exemple : Recherche dans un tableau

```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```

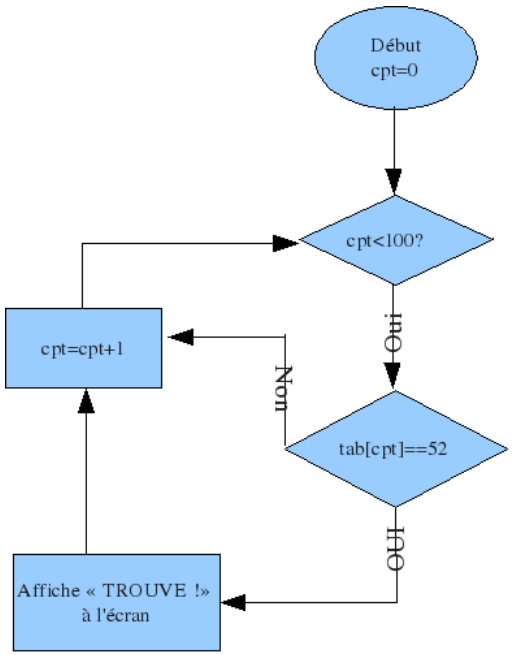
cpt	**0**	**1**	**2**	**3**	**4**	**...**	**99**
tab[cpt]	20	52	-1	52	0	...	45



Exemple : Recherche dans un tableau

```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```

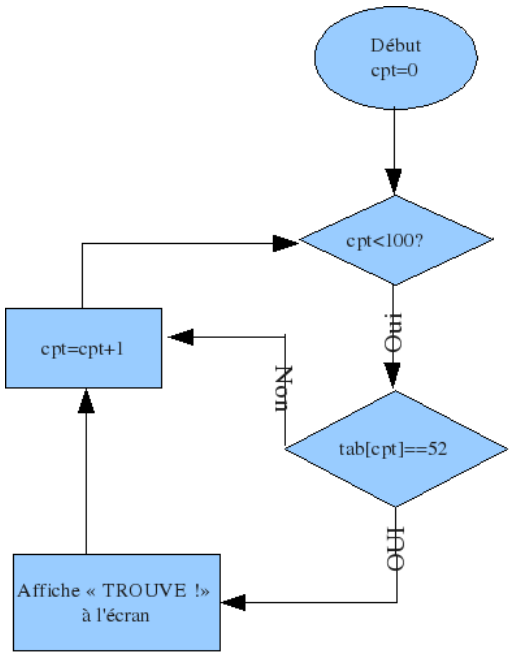
cpt	**0**	**1**	**2**	**3**	**4**	**...**	**99**
tab[cpt]	20	52	-1	52	0	...	45



Exemple : Recherche dans un tableau

```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```

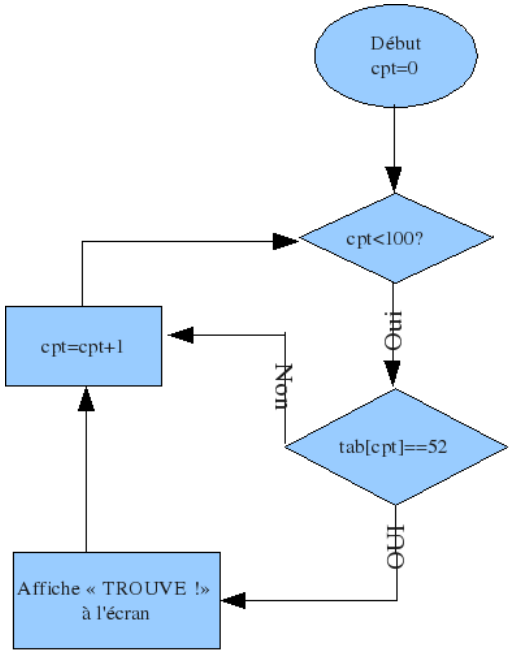
cpt	**0**	**1**	**2**	**3**	**4**	**...**	**99**
tab[cpt]	20	52	-1	52	0	...	45



Exemple : Recherche dans un tableau

```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```

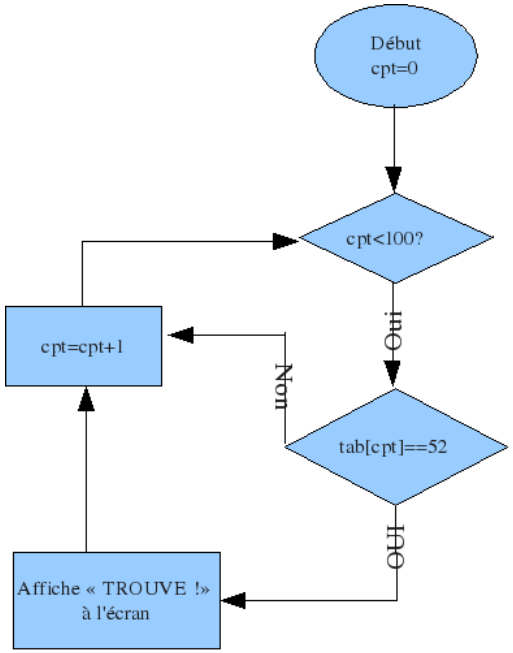
cpt	0	1	2	3	4	...	99
tab[cpt]	20	52	-1	52	0	...	45



Exemple : Recherche dans un tableau

```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```

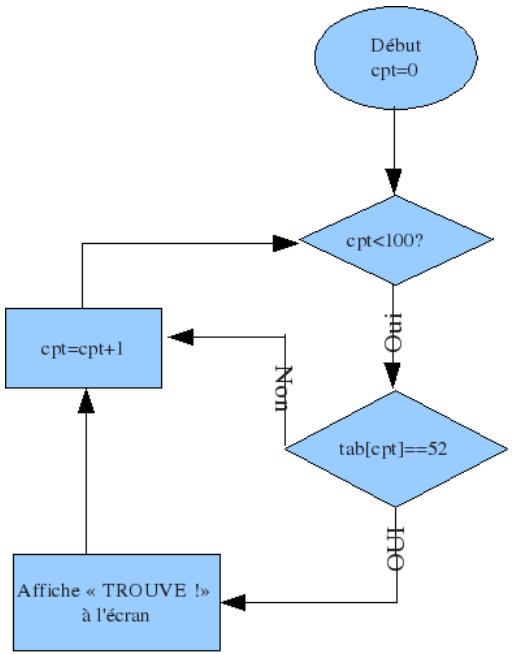
cpt	**0**	**1**	**2**	**3**	**4**	**...**	**99**
tab[cpt]	20	52	-1	52	0	...	45



Exemple : Recherche dans un tableau

```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```

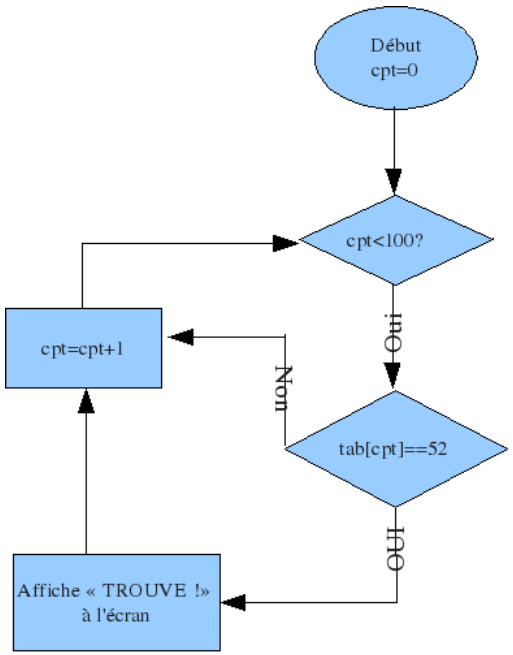
cpt	**0**	**1**	**2**	**3**	**4**	**...**	**99**
tab[cpt]	20	52	-1	52	0	...	45



Exemple : Recherche dans un tableau

```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```

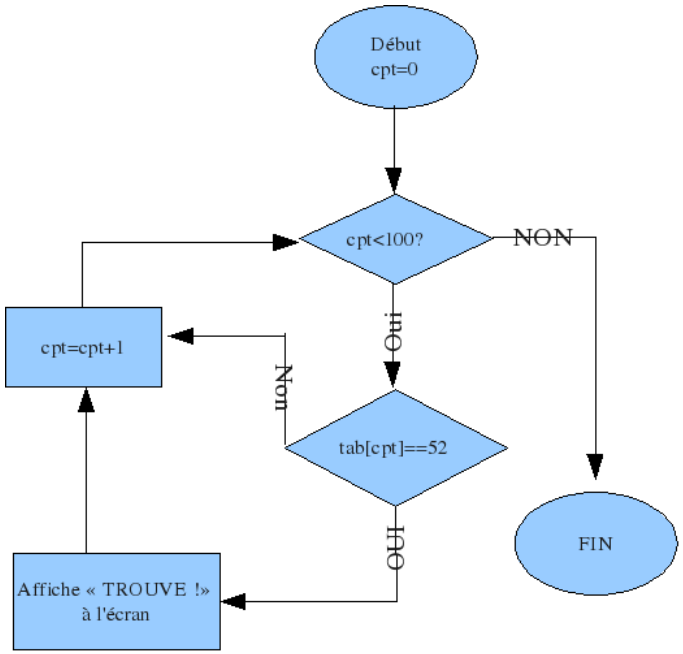
cpt	**0**	**1**	**2**	**3**	**4**	**...**	**99**
tab[cpt]	20	52	-1	52	0	...	45



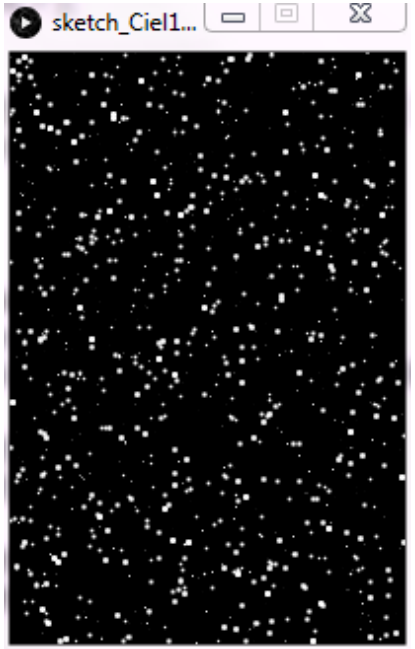
Exemple : Recherche dans un tableau

```
int[] tab = new int[100];
// tab supposé initialisé ici
...
int cpt;
for (cpt=0; cpt<100; cpt++)
{
    if (tab[cpt] == 52)
    {
        println("TROUVE !");
    } // fin du if
} // fin du for (cas où cpt<100 n'est plus vrai)
```

cpt	**0**	**1**	**2**	**3**	**4**	**...**	**99**
tab[cpt]	20	52	-1	52	0	...	45



Exemple : Affichage d'un ciel étoilé




```
1  size(200,300);
2  float[] x = new float[1000];
3  float[] y = new float[1000];
4
5  for(int i=0; i<1000; i++)
6  {
7    x[i] = random(width); y[i] = random(height);
8  }
9
10 int maxT = 3;
11 background(0);
12 stroke(255);
13
14 for(int i=0; i<1000; i++)
15 {
16   strokeWeight(random(maxT)); point(x[i], y[i]);
17 }
```

A vous de jouer !

```
1  println("boucle 0");
2  for (int i=0; i<10; i=i+1)
3  {
4    println("coucou");
5  }
6
7  println("boucle 1");
8  for (int i=0; i<10; i=i+1)
9    println("coucou");
10
11 println("boucle 2");
12 for (int i=0; i<10; i=i+1)
13   println("coucou");
14   println("BYE");
15
16 println("boucle 3");
17 for (int i=0; i<10; i=i+1)
18 {
19   println("coucou");
20   println("BYE");
21 }
```

Que remarquez-vous ?

Conseil concernant les accolades

Conseil (voire Obligation !)

Comme pour l'instruction conditionnelle "if" :

- Utilisez **systématiquement** les accolades pour délimiter vos blocs "for".

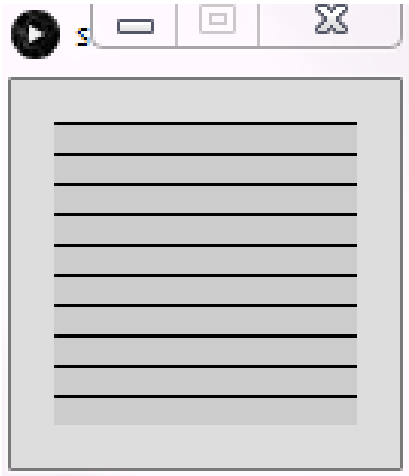
A vous de jouer !

```
println("boucle 4");
for (int i=-1; i<9; i=i+1)
{
  println("coucou");
}

println("boucle 5");
for (int i=0; i<9; i=i+2)
{
  println("coucou");
}

println("boucle 6");
for (int i=10; i>=0; i=i-1)
{
  println("coucou");
}

println("boucle 7");
for (int i=0; i<=9; i=i+1)
{
  println("coucou");
}

println("boucle 8");
for (int i=0; i<9; i--)
{
  println("coucou");
}
```

A vous de jouer !

Voici un programme qui remplit l'écran (100x100) de lignes horizontales espacées de 10 pixels :

```
1  line(0, 0, 100, 0);
2  line(0, 10, 100, 10);
3  line(0, 20, 100, 20);
4  line(0, 30, 100, 30);
5  line(0, 40, 100, 40);
6  line(0, 50, 100, 50);
7  line(0, 60, 100, 60);
8  line(0, 70, 100, 70);
9  line(0, 80, 100, 80);
10 line(0, 90, 100, 90);
```



1. Transformez-le pour l'écrire avec une boucle "for".
2. Rajoutez en tout début size(200,300), et utilisez correctement les variables de Processing width et height de sorte que le programme continue de remplir l'écran de lignes horizontales, quelles que soient les tailles d'écran fournies dans size().

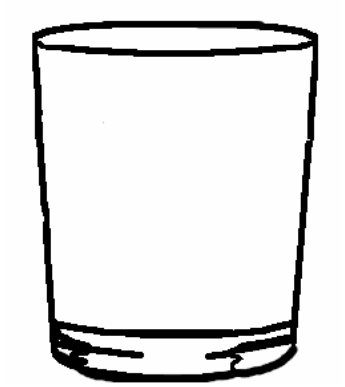
A vous de jouer !

Etudiez les deux codes suivants et effectuez la trace de l'algorithme.
Recopiez, compilez et exécutez le code en mode Debug, le résultat obtenu correspond-t-il à ce que vous aviez prévu ? Concluez.

```
1  int[] tab = new int[5];
2  int i;
3  for (i=0; i<5; i++)
4  {
5    tab[i]=i*2;
6  }
```

```
1  int[] tab = new int[5];
2  int i;
3  for (i=0; i<50; i++)
4  {
5    tab[i]=i*2;
6  }
```

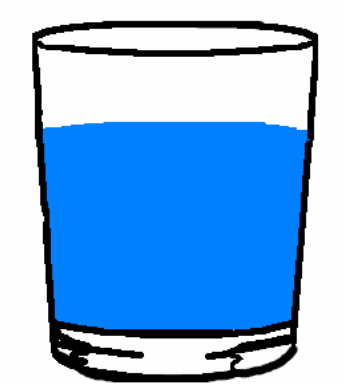
La boucle TANT QUE



Pour remplir un verre d'eau vous utilisez une boucle tant que :

- Ouvrir le robinet
- **Tant que** verre non plein **Faire**
  ne pas toucher au robinet
  Fintantque
- Fermer le robinet

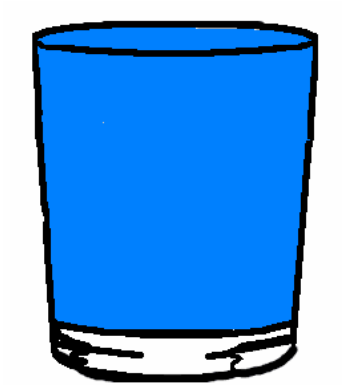
La boucle TANT QUE



Pour remplir un verre d'eau vous utilisez une boucle tant que :

- Ouvrir le robinet
- **Tant que** verre non plein **Faire**
  ne pas toucher au robinet
  Fintantque
- Fermer le robinet

La boucle TANT QUE



Pour remplir un verre d'eau vous utilisez une boucle tant que :

- Ouvrir le robinet
- **Tant que** verre non plein **Faire**
  ne pas toucher au robinet
  Fintantque
- Fermer le robinet

La boucle while

Syntaxe

while (expression booléenne)
{
    Instructions à réaliser dans la boucle
}

Variante

do {
    Instructions à réaliser dans la boucle
}
while (expression booléenne) ;

Exemple

```
1  int[] tab = new int[100];
2  // tab suppose initialise ici
3  //...
4  int cpt = 0;
5  while (cpt<100)
6  {
7    if (tab[cpt]==52)
8    {
9      println("TROUVE !");
10   }
11   cpt = cpt+1;
12 }
```

A vous de jouer

Réalisez le diagramme du code ci-dessus.

Attention

On a toujours le même problème : si le nombre 52 est présent plusieurs fois, la phrase s'affichera plusieurs fois !

Exemple

```
1  int[] tab = new int[100];
2  // tab suppose initialise ici
3  //...
4  int cpt = 0;
5  boolean trouve = false;
6  while ((cpt<100) && (!trouve))
7  {
8    if (tab[cpt]==52)
9    {
10     println("TROUVE !");
11     trouve = true;
12   }
13   cpt = cpt+1;
14 }
```

A vous de jouer

1. Réalisez le diagramme de ce code.
2. Donnez trois exemples de $\{cpt, trouve\}$ permettant d'entrer dans la boucle while.
3. Donnez trois exemples de $\{cpt, trouve\}$ ne permettant pas d'entrer dans la boucle while.

A vous de jouer !

Traduction FOR en WHILE

Reprenez les 2 programmes "Ciel étoilé" et "Lignes horizontales" et transformez les boucles FOR en boucles WHILE.

Prenez du recul

Les transformations précédentes ont-elles permis d'améliorer les programmes ?

- Si oui : pourquoi ? Précisez.
- Si non : pourquoi FOR est mieux adapté que WHILE pour ces programmes ?

Conclusion...

En plus des instructions d'affectation et d'affichage, nous avons maintenant :

- des instructions permettant d'effectuer des **exécutions conditionnelles**,
- des instructions permettant d'effectuer des **itérations**.

Attention

Pour ce qui est des boucles, il faut toujours penser à vérifier :

- début/fin de la boucle : l'initialisation est-elle correcte ? ne va-t-on pas trop loin ?
- incrémentation des compteurs : n'est-on pas dans une boucle infinie ?

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

A vous de jouer !

Mini-Projet 2 : Conditionnelles et Boucles

- Projet commencé en autonomie en séance.
- Sur Moodle, récupérer `miniProjet2.pde` et l'ouvrir dans Processing.
- Pour chaque exercice fait, décommenter son appel dans `setup()`.

Juste avant, un petit mot sur l'instruction `assert`... (page suivante)

Pour la séance TP de la semaine prochaine

- Relire les premiers chapitres.
- Finir les exercices.
- Mini-Projet 2 : terminer les exercices 0 à 7, et les remettre sur Moodle au plus tard en début de séance. Les exercices suivants sont facultatifs.

assert

Pourquoi ?

- L'instruction `assert` permet de **vérifier que "tout va bien"**.
- C'est à destination du développeur, pas de l'utilisateur : ce n'est pas la même chose qu'un `if`.

Comment ?

Simplement en vérifiant qu'un booléen est vrai, à un endroit précis du code. Exemple :

```
1 assert (a - b != 0);
2 c = d / (a - b);
```

- si la condition est *vraie* : on passe à l'instruction suivante.
- si la condition est *fausse* : le programme s'arrête avec une "AssertionError". Bref, il "plante", mais on sait pourquoi...

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

Avant de passer à la suite...

Correction du Mini-Projet 2 : Conditionnelles et Boucles

L'enseignant commente une correction en vidéo-projection.

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

Notion de fonction

Définition

Son intérêt est de diviser un gros problème en une succession de sous-problèmes simples à résoudre.

```
1  ////////////////////////////
2  //FONCTION affiche5Etoiles//
3  ////////////////////////////
4  void affiche5Etoiles()
5  {
6    for (int i=0; i<5; i++)
7    {
8      print("*");
9    }
10   println();
11 }
12
13 ////////////////////////////
14 //FONCTION setup          //
15 ////////////////////////////
16 void setup()
17 {
18   affiche5Etoiles();
19 }
```

Notion de fonction

On applique 1 fois la fonction

affiche5Etoiles()

```
*****
```

A vous de jouer

Exercice

Faites une fonction qui affiche 10 étoiles.

A vous de jouer

Exercice

Faites un programme qui affiche 5 lignes de 5 étoiles.

```
*****
*****
*****
*****
*****
```

Notion de fonction

Solution 1

```
1  ////////////////////////////
2  //FONCTION setup           //
3  ////////////////////////////
4  void setup()
5  {
6    affiche5Etoiles();
7    affiche5Etoiles();
8    affiche5Etoiles();
9    affiche5Etoiles();
10   affiche5Etoiles();
11 }
```

Notion de fonction

Solution 2

```
1  //////////////////////////
2  //FONCTION setup         //
3  //////////////////////////
4  void setup()
5  {
6    for (int i=0; i<5; i++)
7    {
8      affiche5Etoiles();
9    }
10 }
```

Paramètres / Arguments d'une fonction

```
1  ////////////////////////////
2  //FONCTION afficheNEtoiles//
3  ////////////////////////////
4  void afficheNEtoiles(int n)
5  {
6    for (int i=0; i<n; i++)
7    {
8      print("*");
9    }
10   println();
11 }
12 ////////////////////////////
13 //FONCTION setup          //
14 ////////////////////////////
15 void setup()
16 {
17   afficheNEtoiles(8);
18 }
```

Vocabulaire

- La variable `n` est le **paramètre** lors de la définition de la fonction `afficheNEtoiles`.
- Le nombre `8` est l'**argument** lors de l'appel de la fonction `afficheNEtoiles`.

Paramètres / Arguments d'une fonction

```
1  ////////////////////////////
2  //FONCTION afficheNCaracs//
3  ////////////////////////////
4  void afficheNCaracs(int n, char val)
5  {
6    for (int i=0; i<n; i++)
7    {
8      print(val);
9    }
10   println();
11 }
12
13 ////////////////////////////
14 //FONCTION setup          //
15 ////////////////////////////
16 void setup()
17 {
18   afficheNCaracs(8,'a');
19 }
```

Paramètres / Arguments d'une fonction

Exemple

```
aaaaaaaa
```

Attention

afficheNCaracs('a',8) $\neq$ afficheNCaracs(8,'a')

En résumé

On utilise une fonction pour :

- Appliquer plus d'une fois un traitement
- Paramétrer un traitement : celui-ci est réalisé à chaque appel de fonction, avec les valeurs données lors de cet appel
- Structurer des traitements (découper réduit la complexité)
- Échanger clairement des informations entre fonctions : qui fait quoi ? Sur quelles variables ?
- Réutiliser des ensembles d'actions (notion de boîte noire)

En résumé

On utilise une fonction pour :

- Appliquer plus d'une fois un traitement
- Paramétrer un traitement : celui-ci est réalisé à chaque appel de fonction, avec les valeurs données lors de cet appel
- Structurer des traitements (découper réduit la complexité)
- Échanger clairement des informations entre fonctions : qui fait quoi ? Sur quelles variables ?
- Réutiliser des ensembles d'actions (notion de boîte noire)

En résumé

On utilise une fonction pour :

- Appliquer plus d'une fois un traitement
- Paramétrer un traitement : celui-ci est réalisé à chaque appel de fonction, avec les valeurs données lors de cet appel
- Structurer des traitements (découper réduit la complexité)
- Échanger clairement des informations entre fonctions : qui fait quoi ? Sur quelles variables ?
- Réutiliser des ensembles d'actions (notion de boîte noire)

En résumé

On utilise une fonction pour :

- Appliquer plus d'une fois un traitement
- Paramétrer un traitement : celui-ci est réalisé à chaque appel de fonction, avec les valeurs données lors de cet appel
- Structurer des traitements (découper réduit la complexité)
- Échanger clairement des informations entre fonctions : qui fait quoi ? Sur quelles variables ?
- Réutiliser des ensembles d'actions (notion de boîte noire)

En résumé

On utilise une fonction pour :

- Appliquer plus d'une fois un traitement
- Paramétrer un traitement : celui-ci est réalisé à chaque appel de fonction, avec les valeurs données lors de cet appel
- Structurer des traitements (découper réduit la complexité)
- Échanger clairement des informations entre fonctions : qui fait quoi ? Sur quelles variables ?
- Réutiliser des ensembles d'actions (notion de boîte noire)

A vous de jouer !

- Écrire une fonction ne prenant aucun paramètre et qui affiche "bonjour"
- Écrire une fonction qui prend un booléen en paramètre et qui affiche "vrai" si l'argument vaut true et affiche "faux" sinon
- Écrire une fonction qui prend un entier en paramètre et qui écrit autant de fois "bonjour" que le nombre passé en paramètre
- Écrire une fonction qui prend un entier en paramètre et qui affiche le résultat de la division par 2 de cet entier

A vous de jouer !

On peut aussi appeler des fonctions depuis les méthodes à usage graphique (draw(), mousePressed(), etc...)

Exercice

- Écrire une fonction ne prenant aucun paramètre et qui affiche dans la console les coordonnées de la souris sous la forme : X= ... Y= ...
- Appeler cette fonction lorsque l'on presse un bouton de la souris

Les fonctions : la sortie

Définition

Les fonctions sont des sous-programmes admettant des paramètres. Elles peuvent également retourner UN SEUL résultat (comme les fonctions mathématiques : $y = f(x, w, ...)$)

- Les paramètres d'une fonction sont en nombre fixe (important pour l'appel)
- Une fonction possède un seul type, qui est le type de la valeur retournée
- Généralement le nom d'une fonction est soit un nom (*e.g.* : minimum) soit une question (*e.g.* : estPaire)

Les fonctions : la sortie

Définition

Les fonctions sont des sous-programmes admettant des paramètres. Elles peuvent également retourner UN SEUL résultat (comme les fonctions mathématiques : $y = f(x, w, ...)$)

- Les paramètres d'une fonction sont en nombre fixe (important pour l'appel)
- Une fonction possède un seul type, qui est le type de la valeur retournée
- Généralement le nom d'une fonction est soit un nom (*e.g.* : minimum) soit une question (*e.g.* : estPaire)

Les fonctions : la sortie

Définition

Les fonctions sont des sous-programmes admettant des paramètres. Elles peuvent également retourner UN SEUL résultat (comme les fonctions mathématiques : $y = f(x, w, ...)$)

- Les paramètres d'une fonction sont en nombre fixe (important pour l'appel)
- Une fonction possède un seul type, qui est le type de la valeur retournée
- Généralement le nom d'une fonction est soit un nom (*e.g.* : minimum) soit une question (*e.g.* : estPaire)

A vous de jouer

```
1  int maxi(int a, int b)
2  {
3    int m;
4    if (a>=b)
5    {
6      m = a;
7    }
8    else
9    {
10     m = b;
11   }
12   return m;
13 }
14
15 void setup()
16 {
17   int resultat;
18   resultat = maxi(3, 5);
19   println(resultat);
20 }
```

A vous de jouer [facultatif, pour les plus rapides]

```
float
  x1=200, x2=300, x3=400,
  y1=200, y2=200, y3=200;

float DIAMETRE = 80;

color
  FOND = color(0),
  BLEU = color(0,0,255),
  BLANC = color(255),
  ROUGE = color(255,0,0);

void setup()
{
  size(600,400);
}

void mousePressed()
{
  deplacement();
}

void mouseDragged()
{
  deplacement();
}
```

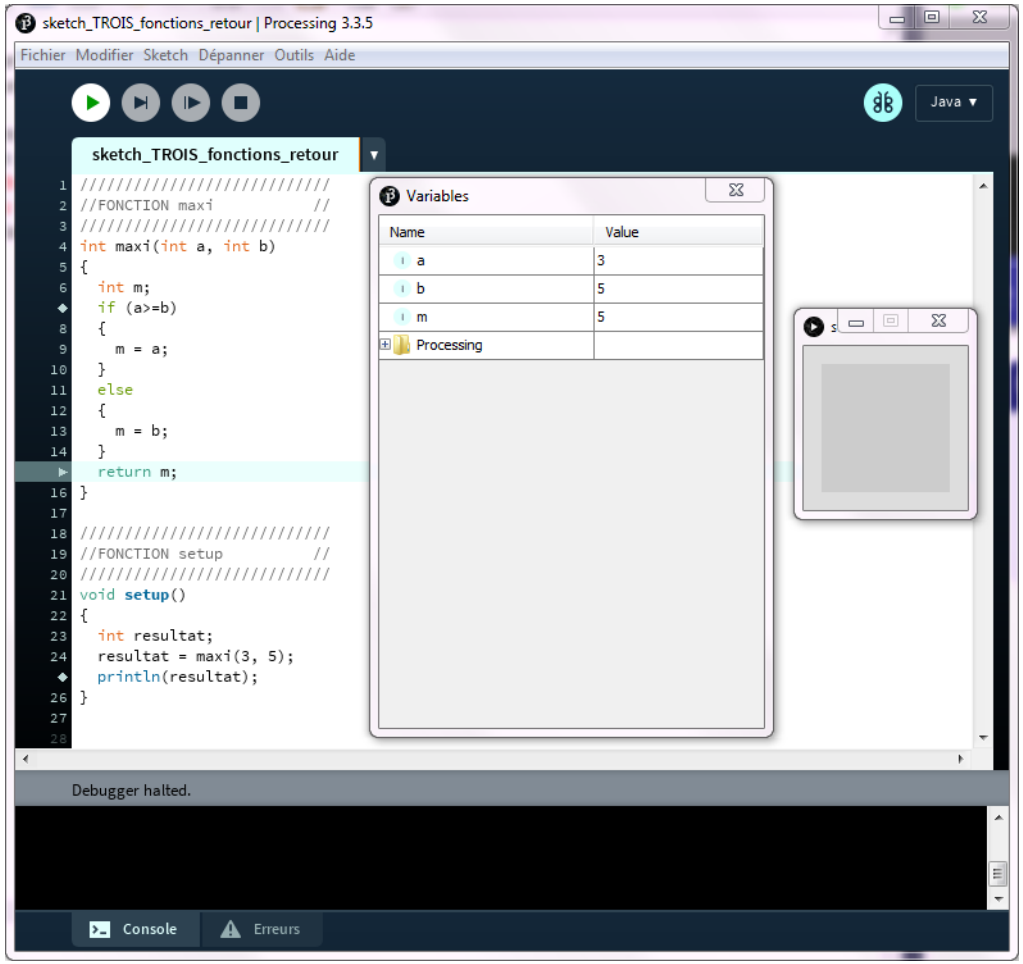
```
void deplacement()
{
  if (mouseButton == LEFT) {
    x1 = mouseX;
    y1 = mouseY;
  } else {
    x3 = mouseX;
    y3 = mouseY;
  }
}

void draw()
{
  background(FOND);
  fill(BLANC);
  text("Clic gauche/clic droit pour deplacer.\n"
  + "Voulu :
bleu, blanc, rouge, de gauche a droite.\n"
  + "Les couleurs doivent changer quand on bouge...",
  20, 20);
  color c1 = BLEU;
  color c2 = BLANC;
  color c3 = ROUGE;
  // TODO : attribuer les bonnes couleurs.
  fill(c1); ellipse(x1, y1, DIAMETRE, DIAMETRE);
  fill(c2); ellipse(x2, y2, DIAMETRE, DIAMETRE);
  fill(c3); ellipse(x3, y3, DIAMETRE, DIAMETRE);
}
```

Le mode Debug de Processing (compléments)

Visibilité des variables :

Dans la fenêtre Variables, on ne voit que les variables locales de la fonction en cours d'exécution.

- Placez un point d'arrêt dans maxi() et un dans setup()
- Exécutez pas à pas et observez

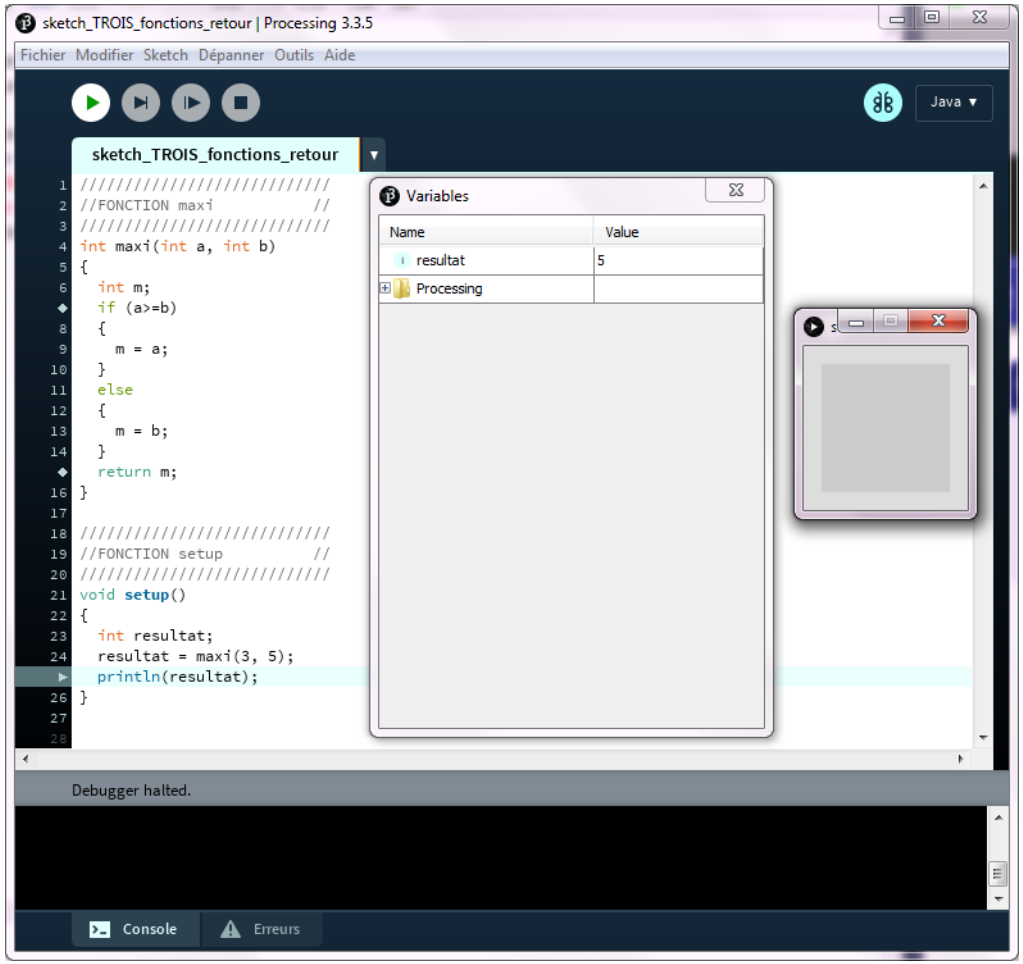


Le mode Debug de Processing (compléments)

Visibilité des variables :

Dans la fenêtre Variables, on ne voit que les variables locales de la fonction en cours d'exécution.

- Placez un point d'arrêt dans maxi() et un dans setup()
- Exécutez pas à pas et observez



A vous de jouer

- Écrire une fonction ne prenant aucun paramètre et qui retourne la valeur 3.14
- Écrire une fonction qui prend un booléen en paramètre et qui retourne l'inverse de ce booléen
- Écrire une fonction qui prend un entier en paramètre et retourne le résultat de la division par 2 de cet entier

Passage de paramètres de fonctions

Uniquement les types de base

Ne sont concernés ici que les types de base : int, float, char, boolean, etc ...

Transmission d'un paramètre par valeur (ou copie)

La transmission d'informations entre **fonction appelante** et **fonction appelée** se fait par valeur (ou copie) :

- la **fonction appelée** travaille sur une variable accessible à elle seule
- elle peut la modifier, mais cela ne peut pas avoir de conséquence sur les variables de la **fonction appelante**

Recopiez le code suivant

```
1  void changer(int first, int b)
2  {
3    int temp;
4    temp=first;
5    first=b;
6    b=temp;
7    println("Dedans : first vaut ", first, " et b vaut ", b);
8  }
9
10 void setup()
11 {
12   int first=3;
13   int last=5;
14   println("Avant : first vaut ", first, " et last vaut ", last);
15   changer(first, last);
16   println("Apres : first vaut ", first, " et last vaut ", last);
17 }
```

La transmission par valeur

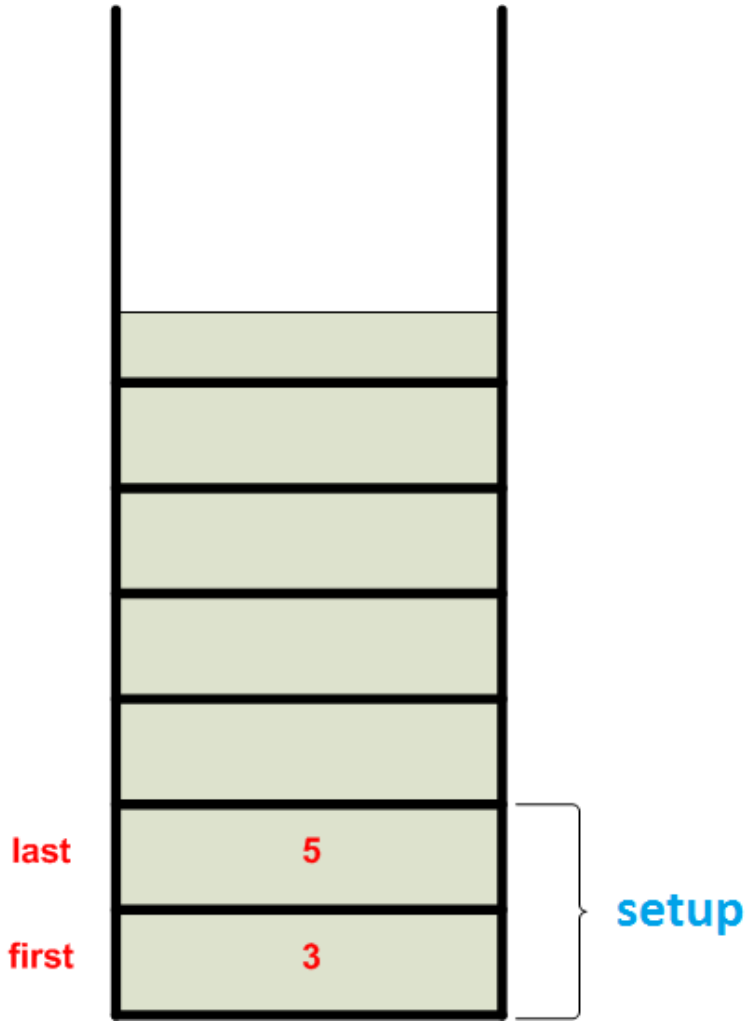
```
1  void changer(int first, int b)
2  {
3    int temp;
4    temp = first;
5    first = b;
6    b = temp;
7    println("Dedans : first vaut ", first, " et b vaut ", b);
8  }
9
10 void setup()
11 {
12   int first = 3;
13   int last = 5;
14   println("Avant : first vaut ", first, " et last vaut ", last);
15   changer(first, last);
16   println("Apres : first vaut ", first, " et last vaut ", last);
17 }
```

```
Avant : first vaut  3 et last vaut 5
Dedans : first vaut 5 et b vaut 3
Apres : first vaut 3 et last vaut 5
```

La transmission par valeur

Exemple

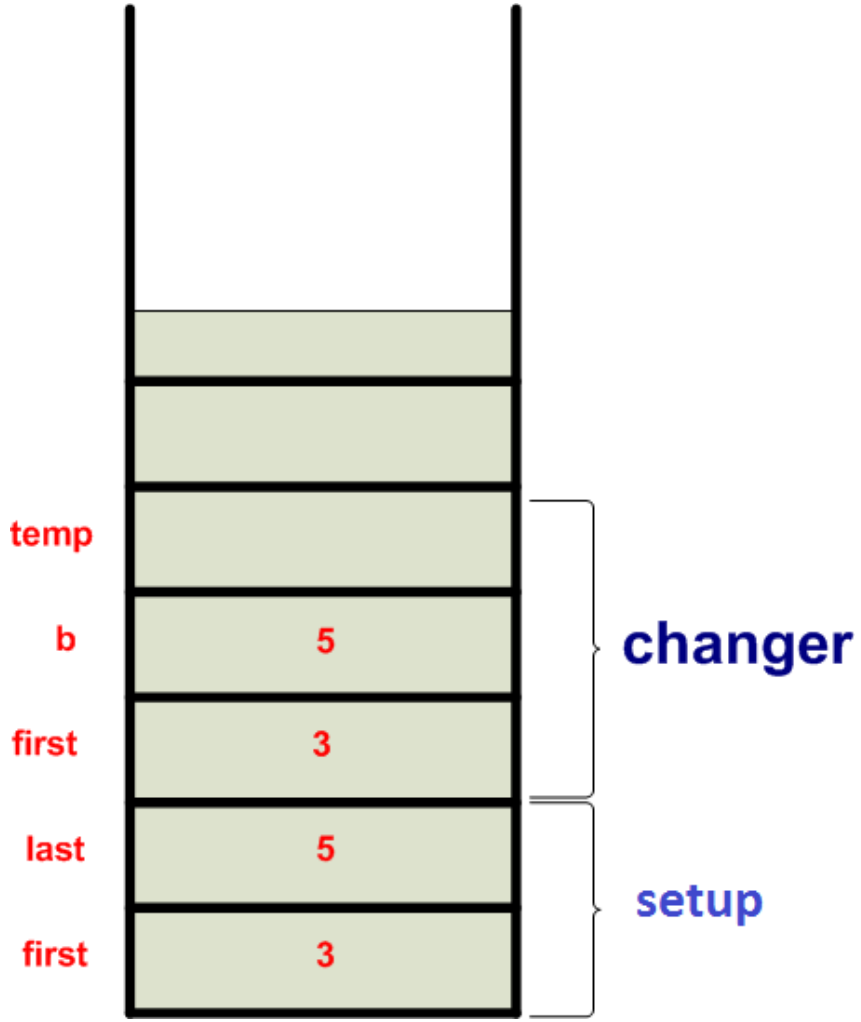
```
void changer(int first, int b)
{
  int temp;
  temp=first;
  first=b;
  b=temp;
  println("Dedans : first vaut ", first, " et b vaut ", b);
}

void setup()
{
  int first=3;
  int last=5;
  println("Avant : first vaut ", first, " et last vaut ", last);
  changer(first, last);
  println("Apres : first vaut ", first, " et last vaut ", last);
}
```



La transmission par valeur

Exemple

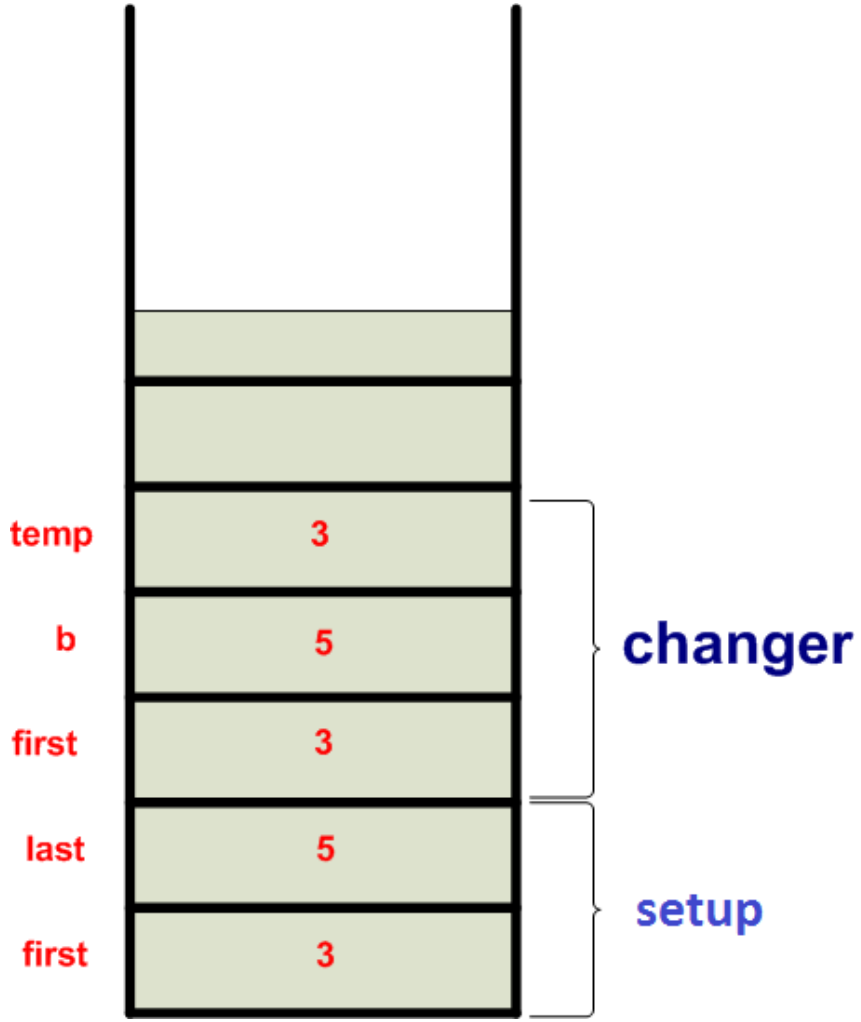
```
void changer(int first, int b)
{
  int temp;
  temp=first;
  first=b;
  b=temp;
  println("Dedans : first vaut ", first, " et b vaut ", b);
}

void setup()
{
  int first=3;
  int last=5;
  println("Avant : first vaut ", first, " et last vaut ", last);
  changer(first, last);
  println("Apres : first vaut ", first, " et last vaut ", last);
}
```



La transmission par valeur

Exemple

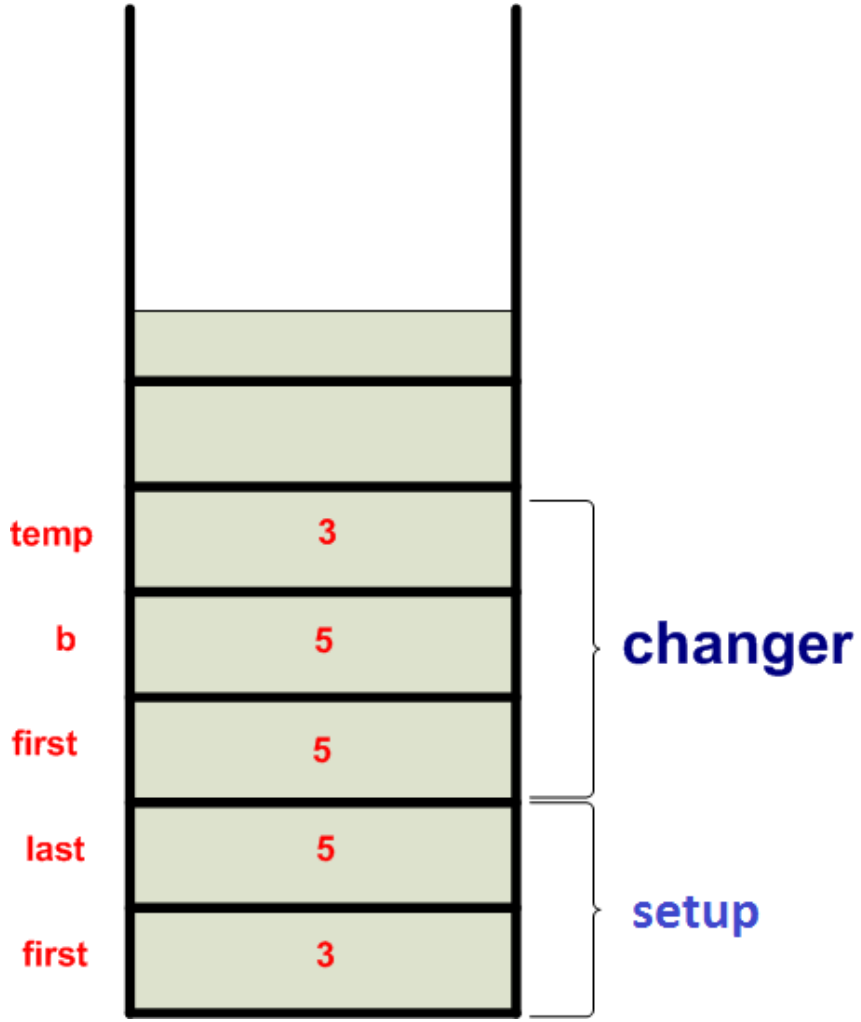
```
void changer(int first, int b)
{
  int temp;
  temp=first;
  first=b;
  b=temp;
  println("Dedans : first vaut ", first, " et b vaut ", b);
}

void setup()
{
  int first=3;
  int last=5;
  println("Avant : first vaut ", first, " et last vaut ", last);
  changer(first, last);
  println("Apres : first vaut ", first, " et last vaut ", last);
}
```



La transmission par valeur

Exemple

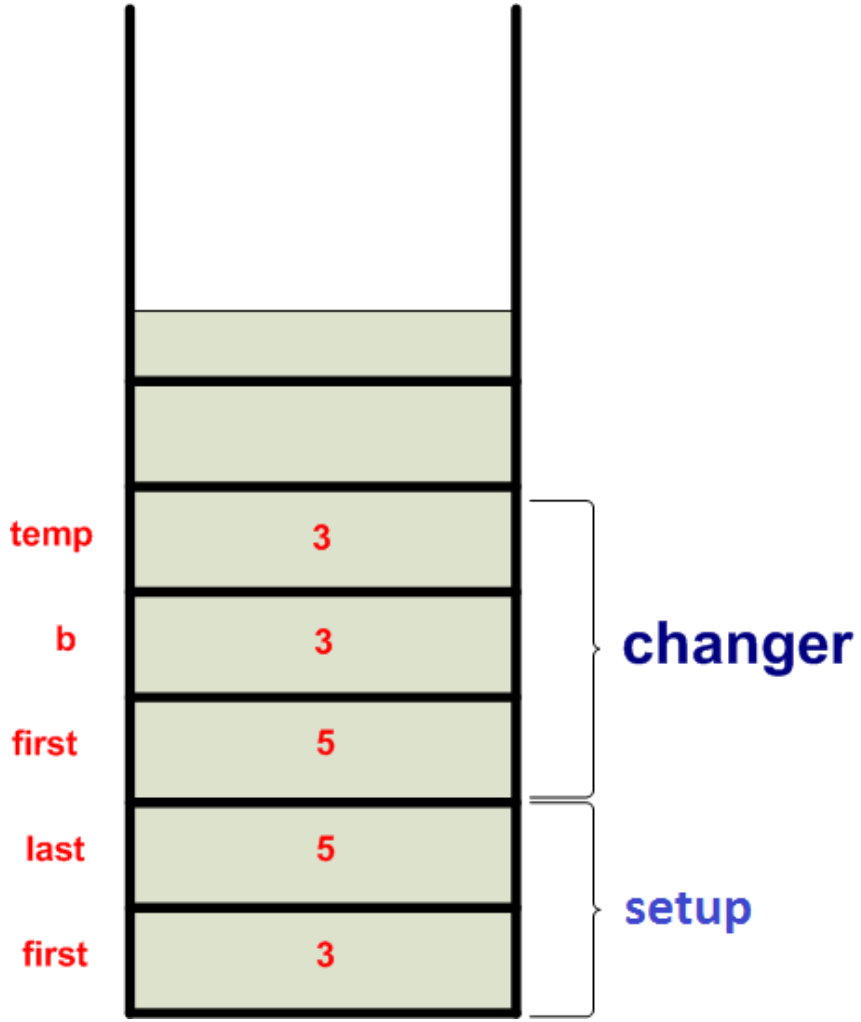
```
void changer(int first, int b)
{
  int temp;
  temp=first;
  first=b;
  b=temp;
  println("Dedans : first vaut ", first, " et b vaut ", b);
}

void setup()
{
  int first=3;
  int last=5;
  println("Avant : first vaut ", first, " et last vaut ", last);
  changer(first, last);
  println("Apres : first vaut ", first, " et last vaut ", last);
}
```



La transmission par valeur

Exemple

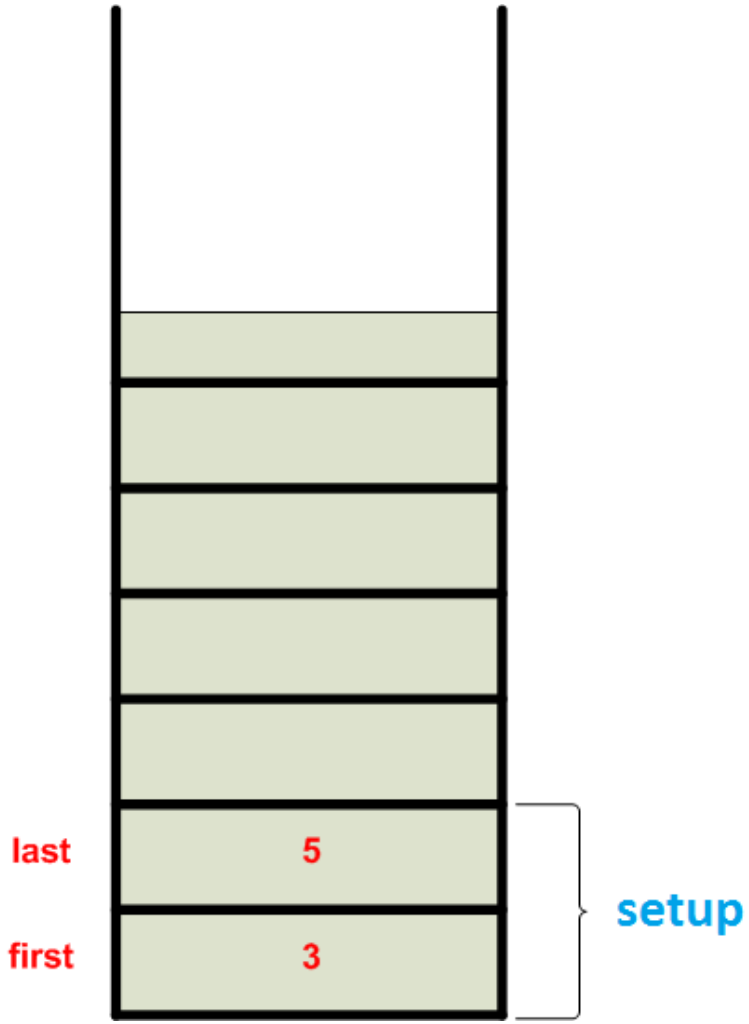
```
void changer(int first, int b)
{
  int temp;
  temp=first;
  first=b;
  b=temp;
  println("Dedans : first vaut ", first, " et b vaut ", b);
}

void setup()
{
  int first=3;
  int last=5;
  println("Avant : first vaut ", first, " et last vaut ", last);
  changer(first, last);
  println("Apres : first vaut ", first, " et last vaut ", last);
}
```



La transmission par valeur

Exemple

```
void changer(int first, int b)
{
  int temp;
  temp=first;
  first=b;
  b=temp;
  println("Dedans : first vaut ", first, " et b vaut ", b);
}

void setup()
{
  int first=3;
  int last=5;
  println("Avant : first vaut ", first, " et last vaut ", last);
  changer(first, last);
  println("Apres : first vaut ", first, " et last vaut ", last);
}
```



A vous de jouer

1. Indiquez ce que donnent les quatre affichages.
2. Dessinez, dans chaque cas, l'état de la pile après exécution de l'instruction ligne 13, 3, 14, 8 et 16.

```
1  void fct1(int a)
2  {
3    a=3;
4    println(a);
5  }
6  void fct2(int b)
7  {
8    b=7;
9    println(b);
10 }
11 void setup()
12 {
13   int a=2;
14   fct1(a);
15   println(a);
16   fct2(a);
17   println(a);
18 }
```

A vous de jouer

Que pensez-vous des affirmations suivantes :

- Le nom de la variable dans l'entête de la fonction est important
- Une variable passée en copie est détruite à la fin de l'exécution de la fonction
- Dans fct2 je peux ajouter entre la ligne 7 et 8 l'instruction suivante : `println(a)` ;

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

Tester une fonction

Pourquoi ?

- pour vérifier que le code de la fonction est **correct** !
- parce qu'une fonction est **facile à tester** (code isolé).
- permet de vérifier que l'on a bien compris ce qu'est censée faire la fonction,
- permet de penser aux cas limites, avant même d'écrire le code.

Comment ?

Via des tests unitaires, c'est-à-dire

- des cas de tests appliqués à la fonction,
- pour lesquels on va *vérifier* que la fonction est correcte : ne plante pas, renvoie la bonne valeur, effectue la bonne action. . ..

Tests unitaires

Comment ?

- **via du code dédié** : code qui va lancer la fonction à tester et vérifier qu'elle est correcte.

Techniquement ?

- À chaque fonction que l'on veut tester, on associe une *fonction de test*,
- dans laquelle nous utiliserons `assert` pour vérifier les résultats.

fonction :

```
int maxi(int a, int b)
{
  ...
}
```

fonction de test :

```
void testMaxi()
{
  assert(maxi(3, 7)==7); // un cas de test
  ...
}
```

Tests unitaires

Avantages du code dédié

- permet de lancer facilement tous les tests :
  évite de devoir lancer l'application et de l'utiliser jusqu'à l'appel de la fonction, pour chaque cas de test, à chaque modification du code...
- et donc, de vérifier qu'une modification n'introduit pas un bug (*non-régression*)

Inconvénients du code dédié

Il faut l'écrire...

Cela devient une "bibliothèque de tests" qui vit à côté du code principal, et doit être maintenu.

À vous de jouer

Tests unitaires

1. Reprenons l'exemple de la fonction maxi.

```c
int maxi(int a, int b)
{
  int m;
  if (a<=b)
  {
    m = a;
  }
  else
  {
    m = b;
  }
  return m;
}
```

```c
void testMaxi()
{
  assert(maxi(3, 7)==7);   // cas a<b
  assert(maxi(7, 3)==7);   // cas a>b
  assert(maxi(-7, 3)==3); // cas a<b avec a<0
  assert(maxi(0, -3)==3); // cas a>b avec b<0
  assert(maxi(5, 5)==5);   // cas a=b
}

void setup()
{
  testMaxi();
}
```

2. Compilez et exécutez ce code, et observez qu'une erreur est détectée dans la fonction maxi (utilisez 1 point d'arrêt).
3. Corrigez l'erreur, puis relancez les tests, et ce jusqu'à ce qu'il n'y ait plus d'erreur.

Plan du cours

1. Séance 1
   - Environnement
   - Commandes Processing
   - Variables et Types
   - Mini-Projet 1
2. Séance 2
   - Variables et Types : Tableaux
   - Tests et Boucles
   - Mini-Projet 2
3. Séance 3
   - Fonctions
   - Tests unitaires
   - Mini-Projets 3, 4 et escape game

Mini-Projet 3 : Jeu des allumettes

Projet réalisé en autonomie en séance.

Découpage fonctionnel, documentation et tests unitaires

- Récupérez les fichiers :
  - `sujetAllumettes.txt` : le sujet,
  - `jeuAllumettes.pde` : permet de jouer au jeu des allumettes,
  - `jeuAllumettesAModifier.pde` : dans lequel vous écrirez vos réponses.
- Partie 1 : restructuration du code
- Partie 2 : rédaction de la documentation
- Partie 3 : rédaction de tests unitaires

À remettre à la fin de la séance

Remettre sur Moodle en fin de séance le fichier `jeuAllumettesAModifier.pde`, quel que soit l'état d'avancement.

Mini-Projet 4 : Fonctions mystères

Projet réalisé en autonomie avant le TP suivant.

Appels de fonctions, documentation et tests unitaires

Récupérez le fichier `fonctionsMysteres.pde`. Il contient l'énnoncé. C'est également dans ce fichier que vous écrirez vos réponses.

À remettre avant le prochain TP

Remettre sur Moodle avant le TP de la semaine prochaine le fichier `fonctionsMysteres.pde`, quel que soit l'état d'avancement.

Mini-Projet bonus : Escape Game

Ce projet n'est pas à rendre, il est facultatif.

Exploration...

crédit photo : Kim Daram

Vous êtes coincé dans le bureau du directeur.
Arriverez-vous à en sortir en moins de 15 minutes ?

Récupérez l'archive `escapeGame`. Ouvrez dans Processing le fichier `escapeGame.pde`. Exécutez-le puis essayez de vous échapper.